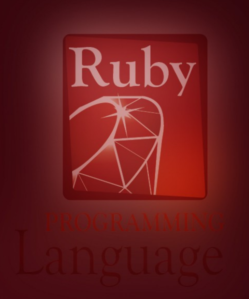


مقدمة في





2008

فهرس المحتويات

مقدمة

لماذا هذا الكتاب ؟

السبب الرئيسى هو أنه كتاب مجانى ومتكامل، لاأعتقد أنك بعد قراءته ستواجه صعوبة فى التعامل مع Ruby

المتطلبات

كمبيوتر ويفضل عليه إحدى نظم GNU/LINUX أو Windows XP
معرفة كيفية إستخدام Text Editor
معرفة جيدة بالصلاحيات Permissions وال Users/Groups لأنى لن أتعرض لها
--تستطيع الحصول على دعم متكامل فى اللينكس من خلال مجتمع لينكس العربى

الإصدارات القادمة والتحديثات

آخر إصدار سيوجد دائما على الموقع الرسمى
Programming-Fr34ks.NET
الإصدار القادم سيشمل التعامل مع المكتبة الرسومية QT4 و مكتبة Wx

المساهمين

أحمد - كرستيان

حقوق النسخ:

الاقتراحات والانتقادات يرجى إرسالها على guru.python[at]gmail.com

الفصل الأول : ماذا ، و متى ولماذا؟

ماهي البرمجة ؟

إذا كنت تكتب برنامج ما فذلك يعني أنك تعطي تعليمات للحاسوب لينفذها .. بمعنى أبسط إنك تملك المقدرة على مخاطبة الحاسوب.

مامعنى التنقيح ؟

التنقيح (Debugging) هى عملية تصحيح للأخطاء الموجودة بمصدر برنامجك .. لأنك ربما تخاطب الحاسوب بطريقة خاطئة فلن يفهم ما تريده بالضبط .. أو يمكن أن ينفذ بطريقة خاطئة .

ماهي Ruby ؟

هى لغة برمجة يابانية المنشأ كتبها Yukihiro Matsumoto ومشهور ب Matz ، سبب التسمية جاء من اسم أحد الأحجار الكريمة -مش روبى بتاعتنا :D -

بدأ فيها عام 1993 وظل محتفظ بها لنفسه حتى 1995 .. حتى اطلقها للعالم

لماذا Ruby ؟

تتميز بـ :

1- مفتوحة المصدر
2- ذات أغراض عامة
3- كائنية المنحى
4- ديناميكية ، و مفسرة
5- محمولة
6- صيغة نظيفة

مفتوحة المصدر : كون اللغة مفتوحة المصدر يعنى أنه يوجد عدد غير منتهى من المطورين قائمين على اللغة، وسرعة فى معالجة الأخطاء .

ذات أغراض عامة : يعنى إمكانية إستخدامها فى أنواع مختلفة من البرامج سواء في قواعد البيانات أو واجهات رسومية أو برامج علمية أو الويب ...إلخ.

لغة كائنية المنحى : الميزة هى أنها الأسلوب الأفضل والأمن لتطوير البرمجيات

ديناميكية: لن تكون مضطرا أن تعلن عن نوع المتغير، بعكس لغات ثانية مثل ال Java مثلا :

```
int i=5; //ساكنة
i=5 //ديناميكية
```

مفسرة: بمعنى أنها تستخدم مفسر وليس مترجم ، سيتبادر إلى ذهنك ما هو المفسر و المترجم.

المترجم هو برنامج يقوم بتحويل الكود من لغة مثل السى مثلا إلى ملف تنفيذي exe
المفسر هو برنامج يقوم بتنفيذ الكود سطر بعد سطر وهذا له مميزات و سلبيات:
فمن المميزات: هى المحمولية على أكثر من نظام تشغيل وأكثر من بنية من العتاد.
بعكس اللغات المترجمة التي ستحتاج إلى إعادة ترجمة برنامجك كل مرة لكل منصة.
من االسلبيات : البطء و إمكانية الإطلاع على الكود -وهذه لا تعتبر مشكلة في عالم المصادر الحرة-
المحمولية : بمعنى أنها مدعومة على العديد من النظم كما ذكرنا في الجزء السابق.

تثبيت روبي

لاحظ ان لغة روبي مثلها مثل العديد من اللغات المكتوبة باللغة السي فهي تستخدم حزمة جاهزة او عليك بتحميل مصدر الكودي للغة المكتوب بلغة السي و ترجمته على جهازك.

على كل سنشرح الطريقتين:

اولا تثبيت الحزم الجاهزة:

لاحظ أن روبي غالبا تأتي مثبتة مع توزيعة اللينكس التي معك.
في التوزيعات المشابهة لردهات:
اعمل استعلام ، هل روبي مثبتة على نظامك أم لا ، باستخدام الأمر التالي:

```
rpm -q ruby
```

إذا لم تكن مثبتة استخدم yum كمستخدم جذر ، و استخدم الأمر التالي:

```
yum install ruby
```

في التوزيعات المشابهة لدبيان:

إذا لم تكن مثبتة استخدم apt-get

```
sudo apt-get install ruby
```

على نظام الويندوز:

نزل حزمة روبي من الموقع الرسمي لنظام الويندوز:

http://www.ruby-lang.org/en/downloads/

افتح ال cmd واكتب ruby -v لمعرفة الإصدار

```
C:\Documents and Settings\StrikerX>ruby -v
ruby 1.8.6 (2007-03-13 patchlevel 0) [i386-mswin32]
```

اكتب fxri وهو واجهة رسومية للتوثيق وفيه interactive ruby shell

بيئات التطوير

توجد بيئات تطويرية كثيرة لـ Ruby/Rails

أنا أفضل استخدام بيئة التطويرية: NetBeans 6

يليها Eclipse مع ال RDT plugin

او تقدر بكل بساطة تستخدم Easy Eclipse for Ruby

توجد محررات كثيرة، ومن أشهرها: vim المفضل عندى أو TextMate "جميع فريق ريلز بيستخدمه" ، على كل حال انا سأستخدم vim في كل الكتاب ، ما عدا الفصل الأخير.

ملحوظة : إختيار ال Editor/IDE شئ مهم فى حياة المبرمج .. لأنك لازم تستخدم شئ يساعدك في مشوارك في عالم البرمجة، ومن أهم الأشياء التي يجب أن تتوفر في المحرر أو بيئة التطوير: ال Syntax highlighting وال line numbering وال code folding, .. الخ

للأطلاع على كل بيئات التطويرية للغة روبي راجع الرابط التالي:
http://www.rubymatters.com/ruby-ide.shtml

الفصل الثاني: أهلا بالعالم

برنامجك الأول مع Ruby سيكون Hello, World وهى عادة في أي دورة أن يتبدأ به

افتح محررك واحفظ الملف باسم hi.rb ، لاحظ إن rb هو إمتداد سكربتات ruby ، واكتب:

```
puts "Hello, World!"
```

احفظ التعديل

وشغل الملف كالتالي:

```
ruby hi.rb

C:\Documents and Settings\StrikerX>ruby hi.rb
Hi Ruby!

C:\Documents and Settings\StrikerX>hi.rb
Hi Ruby!
```

ماذا تعني puts ؟

puts هي دالة تستخدم فى طباعة ال strings -سلاسل نصية-

ملاحظة: إذا لم يتعرف نظامك على ruby فكل ما عليك فعله هو إخباره بالمسار الصحيح عندك كالتال:

```
set path=%path%;RUBY_PATH\bin
```

حيث RUBY_PATH هو المسار الذي ثبت فيه Ruby

أو كالتالي:

1)انقر باليمين على My Computer ثم Properties

2) اختر Advanced Tab

3)ثم Environment Variables

4)في Variables لـ (UserName) : على PATH ثم اختر Edit

5)أضف RUBY_PATH;

لا تنس إضافة الفاصلة المنقوطة (;)

سنتعامل فى معظم الوقت مع Ruby بطريقة التفاعلية، بمعنى أن السطر الذي ستكتبه سينفذ في وقته، مثال ذلك:

```
C:\Documents and Settings\StrikerX>ruby -e "puts 'Hello!'"
Hello!
```

لاحظ إن ال e هى switch معناها execute وهى بتنفذ العبارة التي بعدها ولكنه أسلوب شاق قليلا، لذا سنستخدم irb script أو تقدر تستخدم المتوفرة فى ال fxri

فى ال cmd اكتب

irb -v

```
C:\Documents and Settings\StrikerX>irb -v
irb 0.9.5(05/04/13)
```

إذا لم يكن موجودا عندك فثبته باستخدام yum او apt-get حسب توزيعتك

```
yum install irb

sudo apt-get install irb
```

```
irb(main):001:0> puts "Hello, RUBY!"
Hello, RUBY!
=> nil#
```

لاحظ كلمة nil وهى تعي أن تعبيرك لم يرجع أي قيمة .. ستتعرف عليه اكثر لاحقا

nil = null = None

روبي كحاسبة

تستطيع إستخدام روبي كحاسبة، وتنفيذ عمليات حسابية

```
irb(main):002:0> 1+2
=> 3
irb(main):003:0> 8*9
=> 72
irb(main):004:0> 10/2
=> 5
irb(main):005:0> 6-9
=> -3
```

مرحبا روبي بشكل أفضل!

أي سكربت لما تكتبه فى روبي أو أي لغة نصية مثل بايثون و بيرل ، يفترض أن يكون السطر الأول هو مسار المفسر الخاص باللغة.

كيف ستعرف المسار ؟

بإستخدام which

```
%>which ruby
/usr/bin/ruby
```

افتح محررك المفضل واكتب

```
#!usr/bin/ruby
puts “Hi, Ruby!”
```

احفظ الملف ك hi2.rb

وشغله كالتالى

```
%>./hi2.rb
```

سيظهر لك الرسالة التالي : Permission denied

فواضح أنه بحاجة لصلاحيات التنفيذ استخدام chmod لإعطاء الصلاحيات اللازمة:

```
%> chmod 755 hi2.rb

%> ./hi2.rb
Hi,Ruby!
```

التعليقات

تخيل أنك كتبت برنامجا مثلا 2000 سطر وجئت لتراجعه بعد سنة واكتشفت إنك نسيت لماذا استخدمته، ولماذا عملت كذا فى الكود ؟ إذا كنت أنت صاحب الكود الأصلي هكذا، فمابالك بشخص يطالع مصدر برنامجك ؟ بكل بساطة التعليقات هي فن التوضيح لمصدر برنامجك، بإن تضيف تلميحات وتوضيحات على الجزئيات التى تتوقع أنها غامضة في برنامجك، ولاحظ أن المفسر أو المترجم سيتجاهل التعليقات هذه، ولن ينفذها لأنه يفهم أن هذه التعليقات للمبرمج وللشخص الذي سيقرأ المصدر ولكن ليس له هو.

مرحبا روبي بشكل أفضل!

افتح محررك المفضل واكتب:

```
#!usr/bin/ruby

puts “Hi, Ruby!”  #printing 'Hi, Ruby' on the screen!
```

لاحظ أن معظم اللغات النصية أو المفسرة تستخدم العلامة: # فى كتابة التعليقات، وهذا الأسلوب أفضله شخصيا مع أنه توجد لغات تقدم أساليب مختلفة إضافة لـ # ولكني أستخدم العلامة # دائما.

تستطيع أن تجعل التعليق يمتد على كذا سطر .. اذا كنت من مستخدمى C مثلا فستجده مشابها لـ /* */

وفى ruby ويكون هكذا:

begin=

end=

مثال:

```
#!usr/bin/ruby

=begin
Author : ahmed youssef
Language: Ruby
Purpose : printing Hello, World!
=end

puts “Hello, World!”
```

المتغيرات

من الاسم معناها متغيرات أي أنها بتتغير قيمتها خلال برنامجك

على سبيل المثال متغير بإسم String سيشير لـ "Hello, World"

لاحظ أن string التي تحمل اسم Hello, World مخزنة فى مكان معين فى الذاكرة .. ولكي نستدعيها داخل برنامجنا يجب أن نعمل لها اسم مستعار لمكانها حتى نقدر نصل للبيانات المخزنة فيها .. لذا نحن لما أردنا استخدام Hello, World أعطيناها اسم String ، وكلما نستخدم كلمة String خلال برنامجنا سيستدعى Hello, World مباشرة !!

لاحظ المثال التالي:

```
irb(main):004:0> name="Ahmed"
=> "Ahmed"#

irb(main):007:0> puts name
Ahmed
=> nil
```

لاحظ الفرق بين name و "name"

الأولى هى متغير والثانية هى سلسلة نصية:

```
irb(main):008:0> puts name
Ahmed
```

```
=> nil
irb(main):009:0> puts "name"
name
=> nil##
```

لاحظ الأمثلة التالية

```
name=”Ahmed Youssef”
city=”Cairo”
lang=”Ruby”
site=”P.F”
age=66
```

أنشأنا متغيرات بإسم name, city, lang, site, age وأعطينها قيم مناسبة لها

تستطيع أن تعرفهم على سطر واحد كالتالي:

```
name, age=”Ahmed”, 20
```

ملحوظة: لاتسمى متغيراتك بكلمات محجوزة باللغة

كيفية الحصول على نوع المتغير ؟

بكل بساطة Ruby مثلما قلنا سابقا كل شيء فيها عبارة عن Object أو كائن وتوجد أشياء أساسية فى كل كائن منها الطريقة class وهذه ترجع لنا نوع الكائن نفسه .. لاحظ التالي:

```
irb(main):020:0> name="Ahmed"
=> "Ahmed"
irb(main):021:0> name.class
=> String

irb(main):022:0> x=10
=> 10
irb(main):023:0> x.class
=> Fixnum##
```

وهكذا ..

كيفية التحويل من int إلى float

بيتم باستخدام الطريقة to_f وهى إختصار ل to float

```
irb(main):024:0> x=10
=> 10
```

```
irb(main):025:0> f=x.to_f
=> 10.0###
```

الطريقة to_s

وهي طريقة تحول الكائن إلى string (سلسلة نصية)، مثلا لو يوجد كائن كالتالي:
x=5555
الكائن هذا عبارة عن عدد صحيح int عادى ، ولكن نحن نريد تحويله إلى سلسلة نصية ؟ فسنستخدم الطريقة to_s كالتالي:

```
irb(main):026:0> x=5555
=> 5555
irb(main):027:0> s=x.to_s
=> "5555"
irb(main):028:0> x.class
=> Fixnum
irb(main):029:0> s.class
=> String
```

لاحظ بمناسبة التحويل انك إذا مررت المعامل 2 لـ int التي تريد تحويلها ستتحول إلى صيغة الثنائية وإذا مررت المعامل ١٦ ستتحول إلى الصيغة الست عشرية كالتالي:

```
irb(main):030:0> 14.to_s(2) #binary
=> "1110"
irb(main):031:0> 14.to_s(16) #hex
=> "e"
```

الثوابت

هى عبارة عن ثوابت لا تتغير قيمتها طوال برنامجك .. أشهر مثال عليها هو PI = 3.1459
يفضل أن تتبع الإسلوب القياسى فى الكتابة بإن تجعل حروف الثوابت حروف كبيرة.

صفوف الأعداد

Ruby فيها العديد من الصفوف المستخدمة فى التعبير عن الأرقام مثل: Integer و Fixnumو Float و Bignum و Rational لكن نحن لا يهمنا غير ال Integer وال Fixnum و تستطيع أن تطلع على الباقى في التوثيق.

كيف يتم التحويل القسري ؟

على فرض اعندنا float على الشكل هذا 10.8

ونريد أن نحوله إلى integer ، فكل الذي عليك هو تمرر الرقم للصف Integer كالتالي:

```
irb(main):035:0> floatVar=10.8
=> 10.8
irb(main):036:0> intVar=Integer(floatVar)
=> 10
```

لاحظ إن ال int لا يتعامل مع الفاصلة العائمة فيتجاهلها وتبقى القيمة الصحيحة من ال float هي 10

إذا string يمكن أن يكون Integer، على سبيل المثال

"56" نقدر أن نحوله لرقم كالتالي 56

```
irb(main):037:0> Integer("56")
=> 56
```

لكن إذا كان على الشكل التالي مثلا "abc" فلن ينفع، لأن محتوياته ليس أرقام ، بل حروف.

التحويل من Binary

```
irb(main):042:0> Integer (00001110)
=> 584
```

وهكذا ..

succ, next

هي طرق تعيد القيمة التالية للرقم، على الشكل التالي:

```
puts 1.succ #returns 2
puts 5.next #returns 6
```

الفصل الثالث: المتحكمات

If/Elsif/Else

من المؤكد أنك تملك بريدا إلكترونيا .. هل فكرت في كيفية استخدامه ؟ الإجابة نعم، بعد أن أدخل الأسم وكلمة المرور.

افترض أني مقدم خدمة البريد الإلكتروني ، وأنت تسجل الدخول عندي ، تابع هذه الخطوات:

إذا كان اسم المستخدم = شخص ما و كلمة المرور = كلمته السرية ، إذن
أدخله
غير ذلك
لا تسمح له بالدخول

دعنا نطبق الفكرة في مثال:

```
name = "ahmed"
```

إذا المتغير name قيمته = "ahmed" سينفذ الجزئية التي تأتي بعد then .

و ستكون الصورة العامة هكذا:

```
If condition as TRUE then
               if_suite
End
```

دعنا نطبقها:

```
if name==”ahmed” then
             puts “hi ahmed”
end


if 1<2 then
            puts “math genius”
end
```

وتقدر أن تستخدمها كالتالي أيضا:

```
puts "Python time" if name=="StrikerX"
```

لنأخذ مثال بسيط، إذا أردنا أن نعلم طفل أنه إذا كان الطارق الباب يعرفه سيفتح له ، وإن لم يعرفه فلن يفتح له ..

تعالى معنا نطبق مثال على هذا النحو:

```
family="ahmed"
knocker="ahmed"

if knocker==family then
           #open the door
else
           #keep it closed!
```

Unless

هى عبارة تساوي if not (إذا لم يكن)

```
puts "Not 10" unless x== 10

puts "Not 10" if not x== 10
```

سيكون الخرج هكذا:

```
Not 10
Not 10
```

العامل الثلاثي

معظم اللغات بتدعمه وهى short if/else

condition ? then: else

```
irb(main):077:0> name=="ahmed"?  "Python" : "ruby"
=> "Python"
```

مثال على المتحكمات If/Elsif/Else

```

programmer="StrikerX"
if programmer=="StrikerX" then
        val= "Python time"
elsif programmer=="St0rM" then
        val= "C time"
elsif programmer=="MutatioN" then
        val=  "Ruby time"
elseif programmer=="SpAwN" then
        val=  "Perl time"
else
```

```
        val= “Who the hell are you?”
end
```

عبارة Case

أكيد لاحظت المثال السابق لغات مثل Ruby و Java، ..إلخ تقدم طريقة مختصرة لـ falling

```
programmer=”StrikerX”

val= case programmer

when “StrikerX”: “Python time”
when “St0rM” : “C time”
when “MutatioN” : “Ruby time”
when “SpAwN”  :”Perl time”
else  “who the hell are you ?”
end
```

لاحظ المثال التالي:

```
age=100
state= case age
     when 1..5 : “Child”
     when 6..12: “Kid”
     when 13..17: “Teen”
     when 18 .. 40: “Adult”
     when 41..60: “waiting death”
     when 61.. 120: “dead”
else “not human?”
end
```

لاحظ معنى 1..5 بأنها الفترة - المدى– بين 1 إلى 5

الفصل الرابع: الدوال Functions

ماهى الدالة Function ؟ ومالفرق بينها وبين الطريقة Method ؟

الدالة هي مجموعة من التعبيرات ويمكن إستخدامها اكثر من مرة ..
الفرق بينها وبين الطريقة هو مجرد تسمية فقط لأن الطريقة هى هى ال الدالة لكن يطلق عليها ذلك لأنها تكون جزء من كائن :)

مثال .. نحن لسبب ما نريد أن نكتب:

```
This program  is written by: ahmed youssef
```

فى بداية البرنامج مثلا وإذا فى إختيار لل help نضمنها فيه وهكذا .. يعنى سنستخدمها أكثر من مرة جميل ؟
فأفضل حل هو إننا نعمل دالة بتمثل السطر هذا ونقدر أن نستخدمها أكثر من مرة ، وسيبقى علينا التعديل سهل، فبدل أن نعدل فى كل جزء فى البرنامج .. نعدل فى الدالة فقط !

```
#The start
puts "This program is written by: ahmed youssef"

#The help block
puts "This program is written by: ahmed youssef"

#Another block
puts "This program is written by: ahmed youssef"

#The end

puts "This program is written by: ahmed youssef"
```

تخيل هذا، إنه ليس منطقي نهائيا بأن تكرر بهذا الشكل... و زد على ذلك إذا رغبت بأن تعدل جزئية معينة ستضطر بأن تعدل في أكثر من مكان ... أليس كذلك؟

فالحل المثالى هو أن تعمل دالة تعبر عن مجموعة من الأوامر التي تريد أن تنفذها، مثلا:

```
def about()
  puts "This program is written by: ahmed youssef"
end
```

فتخيل الكود سيتحول للصورة التالية .. فإذا احتجت أن تعدل فى مجموعة الأوامر ، فإن ستعدل على مجموعة الأوامر الموجود فى الدالة:

```
#The start
about()

#The help block
about()

#Another block
about()

#The end

about()
```

نحن اتفقنا إن الدالة نجعلها تنفذ مجموعة من الأوامر في كل مرة نستدعيها ..

مثلا نريد أن نعمل دالة تطبع لنا كلمة 3 Hi مرات

```
def sayHi3Times()
     3.times do
       puts "Hi"
     end
end
```

لاحظ 3.times سنتعرض لها لما نشرح ال times ، ولكن إفهمها مجموعة الأوامر التي تأتي بعدها نريدها أن تنفذ 3 مرات .. إذا استصعبتها، حاليا حول الكود للتالي:

```
def sayHi3Times()
       puts "Hi"
       puts "Hi"
       puts "Hi"
end
```

إستدعينا الدالة ل 3 مرات مثلا

```
sayHi3Times

sayHi3Times

sayHi3Times
```

سينفذ الأوامر التي تحويها الدالة 3 مرات:

```
Output:

Hi
Hi
Hi
Hi
Hi
Hi
Hi
```

```
Hi
Hi
```

جميل .. لحد الآن كل الذي تعرفه عن الدوال خطأ 100% :D

الذي تعاملت معه يسمى إجراء Procedure – مبرمجين Pascal بكل تأكيد يفهمونني-
الإجراء هو مجموعة من الأوامر تنفذ ولكن ليس لها عائد return !
ولكن ماذا تعني return أو العائد :S ؟

```
def areaOfSquare
  side=4
  area=side*side
  return area #The value returned from the function.
end

theArea=areaOfSquare  #area
puts theArea
```

لاحظ إن مجموعة الأوامر الخاصة بالدالة areaOfSquare يرجع قيمة ، وهى قيمة ال area
لاحظ إننا استطعنا أن نسند الدالة لـ theArea .. لكن الذي حصل هو أننا أسندنا القيمة العائدة من الدالة - ال returned value – إلى theArea

على كل، لما تكلم الناس كلمهم بكلمة الدالة، ولا تقحم كلمة إجراء إلا إذا احتاجها فعلا :)
فى بعض اللغات التي لا تفرق بين الإجراء والدالة مثل ال C/Ruby .. إلخ فيعتبروا أن الإجراء ماهو إلا دالة ولكن ليس لها عائد.

لاحظ مشكلة بالكود أو قل شيء ليس جيد، وهو أننا فارضين أن طول side فى المربع ب 4، فنحن نريد أن نحدد قيمة ال Side لأنه ليس من المعقول أن نكتب مليون دالة فيها قيمة side مختلفة.أليس كذلك ؟

```
def areaOfSquare(side)
  area=side*side
  return area #The value returned from the function.
end

theArea=areaOfSquare(5)  #area
puts theArea
```

اى حاجة تريد أن تتحكم في قيمتها وهى بتسند للدالة اجعلها Parameter وهنا ال parameter هو ال side

لاحظ في:

```
theArea=areaOfSquare(5)
```

ال 5 هى قيمة ال side ولكن هنا إسمها argument !

ماذا Parameter و Argument ؟
بكل بساطة ال Parameter هى القيمة التي تحدد أنت وترسل إلى الدالة، لكن ال Argument هى القيمة اللى أرسلت إلى الدالة ، بمعنى أن ال parameter هو side
وال Argument هى 5

الدوال المبنية

وهى دوال تقدمها لك اللغة نفسها ، وفي حالتنا الآ هى Ruby
وهى عبارة عن أكواد جاهزة ، وتستطيع أن تستخدمها مباشرة:

```
print("Enter your name: ") #Builtin-Function to display an object (to_s)
name=gets     #Builtin-function to read a string from the standard input.
puts "Hello, "+ name
```

print هى دالة مثل put تستخدم فى طباعة سلسلة نصية على الشاشة ولكن لا تضيف إليها سطر جديد ..

تلميح : جرب الكود السابق مع puts وستفهم قصدى ;)

gets هى دالة ايضا تستخدم فى قراءة string من ال stdin

```
“Hello, “+name
```

تعطينا Hello مدمجة مع ال name .. سنتعرض لل Concatenation بالتفصيل إن شاء الله لاحقا

الفصل الخامس : بنى المعلومات

المصفوفات

تخيل أنك مثلا تكتب برنامجا عن فصلك الذي فى المدرسة، وفيه أسماء كل الطلاب الذين في الفصل .. وعلى افتراض أنهم 30 شخص، هل تتخيل تعريف 30 متغير؟؟

إذا قلت نعم، فإنك ستتعب نفسك :D

الحل يكمن فى المصفوفة، وهى عبارة عن بنية للمعلومات تستخدم فى تخزين مجموعة متغيرات مرتبطة ببعض بدون إنشاء متغيرات لتشمل قيم كل واحدا! .. تابع

أولا: لإنشاء كائن للمصفوفة، يوجد أكثر من طريقة ..

1- إستخدام الطريقة new من ال Array Class وهى تعطينا كائن من ال Array Class

```
ary=Array.new #Create an empty array.
p ary
#output: []
```

2- إسناد مباشر

```
ary=[ ]
p ary
#output: []
```

نرجع ثانية، للطريقة new وهى لها بعض الوظائف الخاصة مثلا ال populating وهى وضع قيم بمجرد إنشاء الكائن:

```
ary=Array.new(4) #4 nils!
p ary
#output: [nil, nil, nil, nil]
```

هنا قلنا للطريقة new إننا نريد كائن من صف المصفوفة، ويكون فيه 4 عناصر .. ولكننا لم نحدد قيم العناصر ، فأخذت nil افتراضيا..

```
ary=Array.new(4){"test"} #array contains 4 elements all are test
p ary
#output: ["test", "test", "test", "test"]
```

هنا قلنا للطريقة new إننا نريد كائن من صف المصفوفة، ويكون فيه 4 عناصر وقيمة كل واحد هى "test"

```
std1=”ahmed”
..
```

```
..
std3
```

فالحل الأفضل هو إنك تضع أسماء الطلاب فى Array كالتالى

```
students=[“ahmed”, “ayman”, “christina”, “rogina”, “wael”, “mostafa”, .... ]
```

لاحظ إن المصفوفة أول عنصر فيها ترتيبه 0

```
students=["ahmed", "christina", "rogina", "Wael", "3amer"]
p students #print the array

#output: ["ahmed", "christina", "rogina", "Wael", "3amer"]
```

```
puts students[0]  طباعة العنصر الأول
#output: ahmed
puts students[0..3] العناصر ال 4 الأولى

#output:
ahmed
christina
rogina
Wael
```

```
puts students[-1] #العنصر الأخير
#output: 3amer

puts "ahmed included!" if students.include?("ahmed")
#output: ahmed included!
```

Include?(obj)

هى طريقة "دالة" ضمن ال array class وترجع return ب true او false ، ومن الآخر معناها هل ال array بتشمل ال object أم لا.

```
puts students.length
#output: 5
```

length طريقة تعيد لنا عدد العناصر الموجودة في المصفوفة

size هى اسم مستعار لـ length

```
puts students.size #alias for length
#output: 5
```

delete_at(index)

هى طريقة تأخد 1 parameter وهو ال index للعنصر اللى تريد أن تحذفه

```
students.delete_at(3) #wael is removed.
p students
```

هنا حذفنا العنصر الذي كان ترتيبه 3 -لاحظ إننا نبدأ الترتيب من 0 -

```
#output:["ahmed", "christina", "rogina", "3amer"]
```

clear هي طريقة تستخدم لحذف كل العناصر الموجودة في المصفوفة

```
students.clear #delete all
p students
 #output: 0
```

empty?

هي طريقة تعيد قيمة بولونية ب true او false في حال إذا كانت المصفوفة لا تحوي على عناصر

```
students.clear
puts "Array is empty" if students.empty?
#output: Array is empty
```

insert(index, object)

طريقة تستخدم في إضافة عنصر لل array عن طريق الفهرس index الذي تحدده كالتالي مثلا

```
students=["ahmed", "christina", "rogina", "Wael", "3amer"]
students.insert(4, "ayman")
p students
#output: ["ahmed", "christina", "rogina", "Wael", "ayman", "3amer"]
```

concat(array)

هي طريقة تستخدم في إضافة array اخرى للحالية كالتالي مثلا

```
students=["ahmed", "christina", "rogina", "wael", "3amer"]
students.concat(["ramy", "mona", "3obaida"])

p students
#["ahmed", "christina", "rogina", "wael", "3amer", "ramy", "mona", "3obaida"]
```

last

هي طريقة تعيد اخر عنصر في المصفوفة

```
students=["ahmed", "christina", "rogina", "Wael", "3amer"]
puts students.last #students[-1]
#output: 3amer
```

لاحظ إنك إذا مررت رقم إلى طريقة last فإنها تعيد إليك آخر العناصر بنفس العدد المّرر وهكذا

```
puts students.last(2)
#output
Wael
```

```
3amer
```

first

هى عكس last تماما وهى تعيد اول عنصر وإذا مررت لها رقم سيعيد لك من اول (الرقم) من المصفوفة

```
puts students.first
#output: ahmed

puts students.first(3)
```

اول ثلاثة

```
#output
ahmed
christina
rogina
```

replace

هى طريقة تستخدم فى إستبدال كل العناصر بالمصفوفة بمصفوفة ثانية

```
langs=["Pascal", "C", "Perl"]
langs.replace(["Ruby", "Python", "C#", "Java"])

p langs
#output: ["Ruby", "Python", "C#", "Java"]
```

reverse

هى طريقة ترجع نسخة من المصفوفة ولكن بترتيب عكسي، انظر المثال:

```
reversedLangs=langs.reverse
p langs
#output:["Ruby", "Python", "C#", "Java"] .. Not Changed!

p reversedLangs
#output: ["Java", "C#", "Python", "Ruby"]
```

اكيد أنك لاحظت أن reverse لم تأثر على المصفوفة الأصلية .. ولكن على فرض أننا نريد أن يحدث التغيير على المصفوفة الأصلية ؟ فالحل هو إننا نستخدم reverse!

ماهى reverse! ؟

اى طريقة تجد فيها علامة التعجب "!" اعرف إنها destructive function يعنى تأثيرها سيكون على الكائن نفسه وليس على نسخته وهنا فى مثالنا، عكست العناصر فى الكائن نفسه وليس على نسخة منه:

```
langs=["Pascal", "C", "Perl"]
langs.reverse!
p langs
#output: ["Perl", "C", "Pascal"]
```

لاحظ المثال التالي للدمج:

```
old=["C", "Pascal", "Fortran"]
new=["Python", "Ruby", "Java"]

all= old+new #or all=old.concat(new)

p all
#output: ["C", "Pascal", "Fortran", "Python", "Ruby", "Java"]

all << "C#" # add c#
p all

#output:["C", "Pascal", "Fortran", "Python", "Ruby", "Java", "C#"]

all.push("Perl") # << perl

p all
#output: ["C", "Pascal", "Fortran", "Python", "Ruby", "Java", "C#", "Perl"]

all.pop #remove the last element.
p all

#output: ["C", "Pascal", "Fortran", "Python", "Ruby", "Java", "C#"]
```

تقدر تستخدمها كمجموعة وتطبق عليها حاجات مثل التقاطع والإتحاد والفرق

```
langs=["C", "Python", "Ruby", "Pascal"]
intrep_langs=["Python", "Ruby", "Groovy"]
```

في الإتحاد سنستخدم عامل or وهو ال |

```
#union
un=langs|intrep_langs
p un
#output: ["C", "Python", "Ruby", "Pascal", "Groovy"]
```

وفي الفرق سنستخدم ال minus -

```
#difference
diff=langs-intrep_langs
p diff
#output: ["C", "Pascal"]
```

وفي التقاطع سنستخدم and operator وهو &

```
#intersection
intSec=langs & intrep_langs
p intSec
#output: ["Python", "Ruby"]
```

uniq

هى طريقة تعيد لنا نسخة من المصفوفة مكونة من العناصر ولكن بدون تكرار

!uniq

هى طريقة تحذف كل التكرارات للعناصر وتجعل ظهوره مرة واحدة فقط

تابع المثال التالى لتوضيح الفكرة

```
ary=[1, 2, 3 ,3, 5, 6, 7, 7, 7, 10]
p ary
uniqAry=ary.uniq

p ary
p uniqAry

ary.uniq!
p ary

#output:
[1, 2, 3, 3, 5, 6, 7, 7, 7, 10]
[1, 2, 3, 3, 5, 6, 7, 7, 7, 10]
[1, 2, 3, 5, 6, 7, 10]
[1, 2, 3, 5, 6, 7, 10]
```

sorting

للترتيب استخدم طريقة sort وهى طريقة تعيد نسخة من المصفوفة ولكن مرتبة

اذا أردت أن يكون التعديل دائما على المصفوفة إستخدم !sort

```
ary=[1, 2, 5,623, 14, 512]
p ary.sort #not changed. just a copy!
#output: [1, 2, 5, 14, 512, 623]

ary.sort! #destructive
p ary
#output: [1, 2, 5, 14, 512, 623]
```

المدى ***Ranges***

المدى هو عبارة عن فترة بتتكون من start, end, step

```
rng1to10=1..10 #10 is included
p rng1to10.to_a  #to_a means to array
#output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

rng1to9= 1...10 #10 isn't included
p rng1to9.to_a
#output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

min

هى طريقة تعيد لنا أصغر عناصر المدى

max

هى طريقة تعيد لنا أكبر عناصر المدى

```
rng=1..10
p rng.to_a

puts rng.min #min
#output 1
puts rng.max #max
#output 10
```

إستخدام المدى بصراحة غالبا يستخدم فى تكوين المصفوفات التي تحوي على عناصر كثيرة، مثلا

1 ل 100

او ال a to z

```
alphabet=('a'..'z').to_a
p alphabet

#output ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n",
"o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"]

words=('abc'..'abk').to_a
p words
["abc", "abd", "abe", "abf", "abg", "abh", "abi", "abj", "abk"]
```

لاحظ أن المدى انبنى بناءا على آخر حرف فعادت لنا المقطع ab + الحروف من a إلى k

include?(obj)

هى طريقة تختبر وجود كائن فى المدى او لا وتعيد ب true او false

Hashes

ال Hash عبارة عن بنى معلومات Data Structure مشهورة جدا فى العديد من اللغات ويمكن أن تكون واجهتها قبل تحت مسمى آخر مثل القاموس Dictionary فى Python او Associative array فى Perl وهكذا

ماهو ال Hash ؟

هو قاموس بالمعنى الحرفى أنت بتحدد ليه المفتاح و القيمة key, value

اولا لإنشاء HashTable سنستخدم طريقة new من ال Hash Class

```
hashTable=Hash.new #Create a new Hash object.
```

او

```
hashTable={ }
```

إضافة keys, values

```
hashObject[key]=value
```

مثال:

```
hashTable["name"]="ahmed"

hashTable["age"] =18
hashTable["sex"] = 'm'

p hashTable
#output: {"name"=>"ahmed", "sex"=>"m", "age"=>18}
```

store(key, value)

او عن طريق إستخدام طريقة store وهى تأخد بارامتراين

```
hash.store("lang", "ruby") #Adding key, value by using store method.
#output:{"name"=>"ahmed", "lang"=>"ruby", "sex"=>"m", "age"=>18}
```

استخدام القيم بال hash

نستطيع الحصول على قيمة key مخزن بال hash بإستخدام ال index او الطريقة fetch

```
hashTable[key] #returns the value
hashTable.fetch(key) #returns the value


hash={"name"=>"ahmed", "sex"=>"m", "age"=>18}

puts "Name: " << hash["name"]
#output: Name: ahmed

puts "sex : " << hash.fetch("sex")
#output: sex : m
```

clear

هي طريقة تستخدم فى حذف كل ال keys/values من ال HashObj

shift

تعيد اول Key/Value على صورة مصفوفة مكونة من عنصرين

```
hash={"name"=>"ahmed", "sex"=>"m", "age"=>18}
ary=hash.shift
p ary
#output: ["name", "ahmed"]
```

invert

هى طريقة تستخدم فى عكس ال hash فتقوم بتحول ال keys إلى values وال values إلى keys

```
hash={"name"=>"ahmed", "sex"=>"m", "age"=>18}

invHash=hash.invert

p invHash
#output: {"m"=>"sex", 18=>"age", "ahmed"=>"name"}

puts invHash["ahmed"]
#output: name
```

has_key?(key) #true/false
has_value?(value)   #true/false

ال has_key هي طريقة تستخدم فى إختبار وجود key بال hash

ال has_value هي طريقة تستخدم فى إختبار وجود value بال hash

```
hash.has_key?("name") #true/false
hash.has_value?(18)   #true/false
```

length

ال length هي طريقة تستخدم فى الحصول على عدد الأزواج فى ال hash

```
hash={"name"=>"ahmed", "sex"=>"m", "age"=>18}
puts hash.length #number of pairs

#output: 3
```

to_a

تستخدم فى تحويل ال hash إلى array

```
hash={"name"=>"ahmed", "sex"=>"m", "age"=>18}
```

```
ary=hash.to_a #convert to array
p ary
#output: [["name", "ahmed"], ["sex", "m"], ["age", 18]]
```

keys

هى طريقة تعيد مصفوفة تشمل كل المفاتيح الموجودة في hash

values

هى طريقة تعيد مصفوفة تشمل كل القيم الموجودة في hash

```
hash={"name"=>"ahmed", "sex"=>"m", "age"=>18}

keys=hash.keys
p keys
#output: ["name", "sex", "age"]

vals=hash.values
p vals
#output: ["ahmed", "m", 18]
```

تحويل Array ل Hash

```
ary=["first", 1, "second", 2, "third", 3]
hash=Hash[*ary]

p hash
#output: {"second"=>2, "first"=>1, "third"=>3}
```

مكافأة : العمل مع الرياضيات

بما إننا سنتكلم عن الرياضيات فنحن سنحتاج إلى الاستعانة Delta Force للرياضيات فى RUBY .. وهى وحدة الرياضيات.

افهم الوحدة module على أنها مجموعة من ال Functions/Constants/Classes - جاهزة او تستطيع أن تكتبها- المرتبطة بجزئية معينة ...

و ruby مثلها مثل العديد من اللغات تسهل على المبرمج عمليات الرياضيات بسهولة من خلال استخدام Module جاهزة وهى Math فيها كل -معظم- الذي يحتاجه المبرمج.

توجد دالة في وحدة الرياضيات إسمها constants وهى تعيد كل أسماء الثواب الموجودة بالوحدة:

```
p Math.constants #retrieve all constants.
#output: ["E", "PI"]
```

قيمة ال PI

```
puts Math::PI
```

```
#output: 3.141592653589793
```

قيمة ال E

```
puts Math::E
#output: 2.718281828459045
```

نحن ندرس الرياضيات باللغة الإنجليزية لذا لا أستطيع أن أترجم الجدول الخاص بالدوال ، ولكن أظن أنه واضح.
يوجد العديد من الطرق الخاصة بالمهام الرياضية وتستطيع أن تطلع عليها من خلال التوثيق.

الفصل السادس: البرمجة كائنية المنحى (OOP)

أساسيات

معظم اللغات تنقسم إلى:

1- لغات برمجية إجرائية

ينقسم البرنامج فيها على هيئة Modules و Functions و Structures زى ال C

2- لغات برمجية كائنية المنحى

وهى تنقسم فيها الأكواد على هيئة Classes/Modules

المهم أن الشخص إذا فهم ال OOP جيدا سيجد أن الطريق مفتوح لكي يتعلم لغات كثيرة مثل: Java/C++/C# وغيرها

تخيل أنك تصمم إنسان Human على الورق

الإنسان له صفات مثل الطول والوزن .. هذه الصفات نسميها صفات Attributes أو حقول Fields فى بعض اللغات .. فى Ruby بتسمى صفة Attribute فتعود على الكلمة!

الإنسان هذا له أفعال أيضا يعملها ، مثل أنه يمشى وينام ويأكل ويشرب!

سنطبق كل الذي قلناه هكذا:

هذا التصميم الرئيسي:

```ruby
class Human
  def initialize

  end
end
```

لاحظ ال initialize هى طريقة خاصة - للباني Constructor- بتعبر عن إنشاء الكائن .. فى حالة الإنسان، تستطيع أن تقول ولادته مثلا :D

> ملاحظة: ستلاحظ أننا ننشيء الكائنات من الصفوف باستخدام طريقة new.. هذه الطريقة تستدعي طريقة initialize والتي بدورها تجهز لنا الكائن من الصف.

في أخرى يجب أن نحددها في الإنسان مثل الإسم والنوع واللون -وليس مجرد إنسان عنصرى :D-

```
class Human
  def initialize(name, sex, color)
    @name =name
    @sex  =sex
    @color=color
  end
end
```

ال @ هي عبارة عن مرجع reference للكائن الحالي، بمعنى أن الكائن الذي سينشيء من الصف يجب أن يكون له اسم، فتكون له هذه العلامة

```
@name=the name of the object
@sex  = the sex of the object
@color = the color of the object
```

جميل لحد الآن .. الصفات التي حددناها من إسم ونوع ولون ، تسمى حقول Fields او صفات Attributes .. ناقص الأفعال Actions او السلوك ، وهي التصرفات التي يعملها الإنسان .. طالما أننا قلنا إنه شئ ينفذ اوفعل فهذا معناه إنه مجموعة من الأوامر أليس كذلك ؟ ويعنى بكل بساطة أن الأفعال هى نفسها دوال ..والمهم أن أي دالة مكتوبة داخل الصف تسمى طريقة Method.

نرجع لصفنا لأنه لم يكتمل بعد ..نحن لم نكتب سوى صفاته، وينقصنا أن نطبق أفعال الإنسان نفسه.
الإنسان يعمل ماذا ؟ ما التصرفات أو الأفعال التى يعملها ؟ ياكل .. يشرب .. ينام .. يشتغل ... إلخ
جيد ، سنعمل تطبيق لهذه الأفعال في صفنا.
سيتحول صفنا للصورة التالية:

```
class Human
  def initialize(name, sex, color)
    @name =name
    @sex  =sex
    @color=color
    @legs=2
    @eyes=2
  end

  def move()
    #moving code
  end

  def sleep()
    #sleeping code
  end

  def drink()
    #drinking code
```

```
  end
end
```

أول شئ هو الباني Constructor وحددنا فيه الصفات التي نريد أن نخصصها للصف .. لاحظ أنه توجد صفات لم نحددها في الباني ولكن هي موجودة بالفعل عند كل الناس بلا إستثناء، فلن يكون لها معنى أن نحددها فى الباني مثل عدد العيون والأرجل.

فكل الحقول أو الصفات المتعلقة بالكائن نسميها متغيرات النسخة Instance Variables لأن كل نسخة لها حقول مختلفة عن الثانية .. وللتمييز بسهولة هى كل الحقول التى فيها @

ثانى شئ السلوك الخاص بصفنا وهو عبارة عن دوال مثل ماقلنا وهى sleep, work, drink, move

جميل ولكن كيف نقدر أن نتعامل مع الحقول التي في الصف ؟

يوجد طريقتين .. الأولى بأن تسمح لأى أحد أن يتحكم فى الحقول وهذا خطأ! لأنه ممكن أن يضع قيم غير منطقية! الطريقة الثانية، بكل بساطة تستطيع أن تستخدم تكنيك ال get/set وتختبر القيم التي سيحاول المستخدم أن يعدل بها الحقول، لاحظ التالى مثلا لل name field

```
  #Getter
  def getName
    return @name
  end

#Setter
  def setName(new_name)
    if new_name.class != string:
      puts "Error!"
      exit(1)
    else
      @name=new_name
    end
  end
```

محاولة لتغيير الإسم بقيمة غير منطقية

```
ahmed=Human.new("ahmed", "m", "white")
ahmed.setName(10)
#ERRORRRRR
```

توجدأساليب أفضل سنتعرض لها إن شاء الله.. ولاحظ أن تستطيع أن تعمل نفس الشئ بالباني Constructor حتى تضمن أن الكائن أنشيء بطريقة صحيحة.

ال getName هى طريقة تعيد لل @name

ال setName هى طريقة تعدل ال @name إلى new_name

مثلا ال sex بتاع ال Object

```
  #Getter
  def getSex
    return @sex
  end
```

سنستخدم Getter فقط وذلك لأننا لا نريد ان نعدل النوع :S

على كل إذا أحببت أن تعدل النوع تسطيع أن تستخدم Setter كالتالى :D

```
  #Setter
  def setSex(new_sex)
    @sex=new_sex
  end
```

فى نوع آخر من المتغييرات غير المتغيرات النسخة وهو المتغيرات الصف وهى عبارة عن متغيرات خاصة بالصف وليست للكائن .. وهى تبدأ ب @@

بكل بساطة أنت لست فاهم!

تابع المثال التالي بتركيز

```
class Human

  @@NUMBER_OF_HUMANS=0
  #The Constructor
  def initialize(name, sex, color)
    @name =name
    @sex  =sex
    @color=color
    @legs=2
    @eyes=2
    @@NUMBER_OF_HUMANS += 1
  end

  def move()
    #moving code
  end

  def sleep()
    #sleeping code
  end

  def drink()
    #drinking code
  end

  def Human.numberOfHumans
    return @@NUMBER_OF_HUMANS
  end
end
```

لاحظ ال @@NUMBER_OF_HUMANS .. المتغير هذا مشترك في كل الكائنات .. بمعنى أن كل الكائنات تستخدم نفس القيمة الموجودة فيه!

```
 def initialize(name, sex, color)
   @name =name
   @sex  =sex
   @color=color
   @legs=2
   @eyes=2
   @@NUMBER_OF_HUMANS += 1
 end
```

لاحظ ان مع إنشاء كائن جديد يوجد متغير فى الصف وهو ال NUM_OF_HUMANS قيمته ستزيد بمقدار 1 وهذا الهدف منه ، إننا نقدر نحسب كم كائن تم إنشاؤه ،وبكل تأكيد الحساب لن يتم عن طريق الكائن ولكن عن طريق الصف !

فسنعرف الطريقة التي ستعيد لنا قيمة ال NUM_OF_HUMANS ولكن نريد أن نجعلها خاصة بالصف فقط ،لذا سنعرفها بالصورة التالية:

```
 def Human.numberOfHumans
   return @@NUMBER_OF_HUMANS
 end
```

لاحظ إننا سبقناها بإسم الصف حتى تفهم Ruby أن الطريقة هذه خاصة بالصف، وهذا النوع من الدوال يطلق عليه الدوال الساكنة static.

جميل جدا .. هكذا أنت فهمت اللعبة، ولكن ينقص شيء واحد فقط و هو أن إسلوب ال Get/Set ليس جميلا :D تخيل الذي سيستخدم صفك، يجب أن يعرف كيف تعمل get و set الحقول .. بالنسبة لى عادى ولكن تخليك تحس أن الكود الذي أمامك شبه فوضوى .. توجد لغات قدمت مفهوم جديد وهو الخصائص Properties وهى عبارة عن تغليف لـ get/set

```
def name
   @name
 end


 def name=(name)
     @name=name
 end

ahmed=Human.new("ahmed", "m", "white")
puts ahmed.name

#output: ahmed
```

```
ahmed.name="youssef"
puts ahmed.name

#output: youssef
```

بسيطة ها ؟

attr_accessor

على كل ، روبي تقدم طريقة أبسط وهي إستخدام ال attr_accessor كالتالي مثلا

```
 attr_accessor  :name

ahmed=Human.new("ahmed", "m", "white")
puts ahmed.name
#output: ahmed

ahmed.name="youssef"
puts ahmed.name
#output: youssef
```

فإذا أحببت أن تعمل getter/setter لأي Attribute فإستخدم attr_accessor!

attr_reader

على فرض إنك تريد أن تجعل الصفة وليكن sex لها get بس ولا تريد أحد أن يغير نوعه :D

فنستخدم attr_reader

```
 attr_reader :sex

ahmed=Human.new("ahmed", "m", "white")
puts ahmed.sex
#output: m


ahmed.sex="f"
#ERROR!
```

attr_writer

على فرض إنك تريد أن تكون للصفة لها set بس فإستخدم attr_writer

Note: لا تستخدم ال attr_writer بدون داعى .. تستطيع أن تعمل set لحقلها من خلال الباني

الوراثة

الوراثة هى احد اهم المفاهيم بال OOP وهى البدأ من حيث إنتهى الآخرون
على فرض إنك تريد أن تصمم صف للموظف ولكن الموظف ماهو إلا إنسان مضاف له بعض الصفات والأفعال التى تميز الموظفين، أليس كذلك ؟

```
class Employer < Human
  def initialize(name, sex, color, salary)
    super(name, sex, color)
    @salary=salary
  end
```

لاحظ إستخدام طريقة super وهى طريقة تنادي الباني الخاص بالصف الأعلى super class وتعمل set للحقول الخاصة به والحقل الخاص بالموظف نفسه وليس للإنسان وهو ال salary سيحدد من ال Employer Constructor

is_a(class)
تختبر هل الكائن هذا مشتق من صف آخر.

instance_of?(class)
هى طريقة تختبر إذا كان الكائن هو كائن من الصف أم لا؟

kind_of?(class)
هي طريقة تختبر إذا كان اصل الكائن هو صف معين أم لا؟

```
h1=Human.new("ahmed", "m", "white")
puts "h1 is an instance of human" if h1.instance_of?(Human) #an Object of the
Human class
puts h1.class #Human

emp1=Employer.new("ahmed", "m", "white", 2000)
puts "emp1 is an instance of Employer class" if emp1.instance_of?(Employer)
puts emp1.class #Employer

puts h1.is_a?(Human) #is h1 a human ?
puts h1.is_a?(Employer) # is h1 an employer ?


puts emp1.is_a?(Employer) # is emp1 an employer?
puts emp1.is_a?(Human) # is emp1 a human ?
puts emp1.instance_of?(Human)
```

```
puts emp1.kind_of?(Human)


#output:
h1 is an instance of human
Human
emp1 is an instance of Employer class
Employer
true
false
true
true
false
true
```

المثال بسيط جدا واعتقد وضح لك الفكرة

جميل .. ال Employer له بعض الأفعال الخاصة به مثل أنه يعمل أو يرفض :D

مثال شامل:

```
class Human


  @@NUMBER_OF_HUMANS=0
  #The Constructor
  def initialize(name, sex, color)
    @name =name
    @sex  =sex
    @color=color
    @legs=2
    @eyes=2
    @@NUMBER_OF_HUMANS += 1
  end

  attr_accessor :name, :color
  attr_reader :sex

  def move()
    #moving code
  end

  def sleep()
    #sleeping code
  end

  def drink()
    #drinking code
  end

  def Human.numberOfHumans
    return @@NUMBER_OF_HUMANS
  end
end
```

```
class Employer < Human
  attr_accessor :salary
  def initialize(name, sex, color, salary)
    super(name, sex, color)
    @salary=salary
    @state=""
  end

  def work
    #working code
  end

  def getHired(salary)
    @salary=salary
    @state="hired"
  end

  def getFired
    @salary=0 #no money :(
    @state="fired"

  end

  def empInfo
    s="Name: " << @name
    s << ", State: " << @state
    s << ", Salary: " << @salary.to_s
    return s
  end

end


class Firm
  def initialize(firm_name, manager)
      @firm_name=firm_name
      @manager=manager
      @emps=[]
  end

  def raiseEmployer(emp, raise)
    emp.salary += raise
    puts @manager + ": " + emp.name + " is raised!\n"
  end

  def hire(emp, salary)
    emp.getHired(salary)
    @emps.push(emp)
    puts @manager + ": " + emp.name + " is hired.\n"
  end

  def fire(emp)
    emp.getFired
    @emps.delete(emp)
```

```
    puts @manager + emp.name + ": " + "is fired.\n"
  end

  def employersList
    return @emps
  end
end


#demo

firm=Firm.new("High-Tech", "Ahmed Youssef")
$ahmed=Employer.new("ahmed", "m", "white", 2000)
$tina =Employer.new("christina", "f", "white", 4000)
$wael= Employer.new("wael", "m", "white", 3000)

def empsInfo
  puts "----------------"
  lst=[$ahmed, $tina, $wael]
  for emp in lst
    puts emp.empInfo
  end
  puts "----------------"
end

puts "Hiring..."

firm.hire($ahmed, 2500)
firm.hire($tina, 2500)
firm.hire($wael, 3000)

empsInfo

puts "Firing wael..."
firm.fire($wael)

empsInfo

firm.raiseEmployer($ahmed, 1000)
empsInfo
puts "--------------"

puts "The list of employers: "
for emp in firm.employersList
  puts emp.name
end
```

الخرج:

```
Hiring...
Ahmed Youssef: ahmed is hired.
Ahmed Youssef: christina is hired.
Ahmed Youssef: wael is hired.
----------------
Name: ahmed, State: hired, Salary: 2500
Name: christina, State: hired, Salary: 2500
```

```
Name: wael, State: hired, Salary: 3000
----------------
Firing wael...
Ahmed Youssefwael: is fired.
----------------
Name: ahmed, State: hired, Salary: 2500
Name: christina, State: hired, Salary: 2500
Name: wael, State: fired, Salary: 0
----------------
Ahmed Youssef: ahmed is raised!
----------------
Name: ahmed, State: hired, Salary: 3500
Name: christina, State: hired, Salary: 2500
Name: wael, State: fired, Salary: 0
----------------
--------------
The list of employers:
ahmed
christina
```

اها بالمناسبة اى متغير يسبقه $ معناه إنه متغير عام تستطيع أن تستخدمه فى أى مكان فى كودك!

هل تذكر لما تكلمنا عن الوحدة Module وقلنا أنها مجموعة صفوف و ثوابت و دوال جاهزة ؟

كل ما عليك هو أنك تأخد كل الذي عملناه من الصفوف لـ empsInfo

وتضعهم في ملف ، وتسمه باسم مرتبط بهم مثلا Business و تضع السطر التالي في بداية الملف

module Business

وهذا في نهايته

end

سيتحول للشكل التالي:

```
module Business
    class Human


  @@NUMBER_OF_HUMANS=0
  #The Constructor
  def initialize(name, sex, color)
    @name =name
    @sex  =sex
    @color=color
    @legs=2
    @eyes=2
    @@NUMBER_OF_HUMANS += 1
  end

  attr_accessor :name, :color
  attr_reader :sex

  def move()
```

```
    #moving code
  end

  def sleep()
    #sleeping code
  end

  def drink()
    #drinking code
  end

  def Human.numberOfHumans
    return @@NUMBER_OF_HUMANS
  end
end


class Employer < Human
  attr_accessor :salary
  def initialize(name, sex, color, salary)
    super(name, sex, color)
    @salary=salary
    @state=""
  end

  def work
    #working code
  end

  def getHired(salary)
    @salary=salary
    @state="hired"
  end

  def getFired
    @salary=0 #no money :(
    @state="fired"

  end

  def empInfo
    s="Name: " << @name
    s << ", State: " << @state
    s << ", Salary: " << @salary.to_s
    return s
  end

end


class Firm
  def initialize(firm_name, manager)
      @firm_name=firm_name
      @manager=manager
      @emps=[]
  end
```

```
  def raiseEmployer(emp, raise)
    emp.salary += raise
    puts @manager + ": " + emp.name + " is raised!\n"
  end

  def hire(emp, salary)
    emp.getHired(salary)
    @emps.push(emp)
    puts @manager + ": " + emp.name + " is hired.\n"
  end

  def fire(emp)
    emp.getFired
    @emps.delete(emp)
    puts @manager + emp.name + ": " + "is fired.\n"
  end

  def employersList
    return @emps
  end
end

def empsInfo
  puts "----------------"
  lst=[$ahmed, $tina, $wael]
  for emp in lst
    puts emp.empInfo
  end
  puts "----------------"
end

def demo

firm=Firm.new("High-Tech", "Ahmed Youssef")
$ahmed=Employer.new("ahmed", "m", "white", 2000)
$tina =Employer.new("christina", "f", "white", 4000)
$wael= Employer.new("wael", "m", "white", 3000)



puts "Hiring..."

firm.hire($ahmed, 2500)
firm.hire($tina, 2500)
firm.hire($wael, 3000)

empsInfo

puts "Firing wael..."
firm.fire($wael)

empsInfo

firm.raiseEmployer($ahmed, 1000)
empsInfo
```

```
puts "-------------"

puts "The list of employers: "
  for emp in firm.employersList
    puts emp.name
  end

  end
end
```

وللإستخدام بكل بساطة اعمل صف يشمل على هذه الوحدة.

الاول تستدعى الوحدة بإستخدام require وتعمل include لها بإستخدام include

```
require "business"

class Dem
  include Business
end

p=Dem.new
p.demo
```

Public/Private/Protected

public: معناها ان الوصول إليها لكل العالم سواء داخل الصف او خارجه

private : معناها أن الوصول إليها لداخل الصف فقط

protected : معناها إن الوصول إليها للصف او للصفوف المشتقة من نفس الصف

```
class A
  def method1

  end

  def method2

  end

  def method3

  end

end

class B < A
```

```
end
```

نعمل كائنات:

```
a=A.new()
b=B.new()
```

لاحظ أن method1, method2, method3 تستطيع أن تستخدمهم من خلال الكائنات A, B وهذا لأنهم Public بشكل افتراضي.

فى ال C# مثلا نحن نعرف أننا نستخدم private حتى نحدد عملية الوصول لإستخدام الطرق سواء للعالم الخارجى او للصفوف التي ستشتق من الصف الحالي.

لكن فى Ruby الوضع يختلف لأن ال private فيها = protected بمعنى أن الطريقة صلاحيات وصولها هي Private تستطيع أن تستخدمها فى الصفوف المشتقة.

لكن دعنا على القاعدة اللى قلناها بالأول

public: معناها ان الوصول إليها لكل العالم سواء داخل الصف او خارجه
private : معناها أن الوصول إليها لداخل الصف فقط
protected : معناها أن الوصول إليها للصف او للصفوف المشتقة من نفس الصف

نضرب مثالا بسيطا
إذا عدلنا المثال السابق إلي

```
class A
  def method1
    puts "method1"
  end

  def method2
    puts "method2 calling method3"
    method3
  end

  def method3
    puts "method3"
  end

  private :method1, :method3
end

class B < A
```

```
    def method4
    puts "method4 calling method3"
    method3
  end
end
```

سننشيء الكائنات:

```
a=A.new()
b=B.new()

b2.method4

#output:
method4 calling method3
method3
```

ستلاحظ أن ال a, b ليس لهم وصول إلى method1, method3 ! لأنهم private تقدر تستخدمهم فى العمليات الداخلية للصف نفسه.

هكذا:

```
  def method4
    puts "method4 calling method3"
    method3
  end
```

فهذا معنى private بكل بساطة.

لن نتكلم عن protected لأن Ruby اهملت مفهومها فجعلتها مثل public :)

تحميل الطرق

وهى إعادة تعريف طريقة موجودة فى الصف الأب أو الصف الأساسي فى الصف الفرعي او الصف الإبن بصورة تتلائم معه

لاحظ المثال:

```
class A

  def method1
    puts "A's method1 is called"
  end

  def method2
    puts "A's method2 calling method3.."
    method3
  end


  def method3
```

```
    puts "A's method3 is called"
  end

  private :method1, :method3


end

class B < A
  def method2 #override the protected method
    puts "B's method2 is called"

  end

  def method4
    puts "Calling method1..."
    method1
  end

end
```

نستخدمهم:

```
a=A.new()
#Calling the parent's method2
a.method2

b=B.new()
#Calling the overriden method2 in the child
b.method2
```

الخرج:

```
A's method2 calling method3..
A's method3 is called
B's method2 is called
```

لاحظ إن method2 في ال B اصبحت مختلفة تماما عنها في ال A وهذا بسبب أنها أعيد تعريفها في ال B

تعدد الأشكال

هذه الكلمة تثير حساسية كثير من الناس مع أن مغزاها سهل جدا وبسيط، وهو أنه يوجد أشكال عديدة من خلال إسم واحد، وسأشرحها حالا

لاحظ المثال التالي:

```
class A
   def method1
     puts "A's method1 is called.."
   end
end

 class B
```

```
  def method1
    puts "B's method2 is called.."
  end

end
```

لاحظ وجود طريقتين بنفس الإسم method1

نعمل objects

```
a=A.new()
b=B.new()
```

نستخدم method1 الموجودة بالإثنين:

```
a.method1
b.method1
```

او بهذه الصورة:

```
objects=[a, b]
for obj in objects
  obj.method1
end
```

فى كلتا الحالتين الخرج سيكون :

```
#output:
#A's method1 is called..
#B's method2 is called..
```

freeze, frozen?

Freeze هي طريقة تمنع الكائن من التعديل عليه

frozen? هي طريقة تستخدم فى إختبار هل الكائن تم تنفيذ freeze عليه أم لا

```
s="Hello"
s.freeze

s << ", World!" #can't be done as "s" is frozen!
puts "freezed!" if s.frozen?
```

to_s

هى طريقة تعبر عن الكائن فى حالة إستخدام puts معه "التمثيل النصي“

to_i

هى طريقة تعبر عن الكائن فى حالة محاولة تحويله ل int

to_f

هى طريقة تعبر عن الكائن فى حالة تحويله ل float

to_a

هى طريقة تعبر عن الكائن فى حالة تحويله ل array

to_hash

هى طريقة تعبر عن الكائن فى حالة تحويله ل hash

الفصل السابع: التكرار

times

times تستخدم فى تكرار block من ال code اكثر من مرة

```
5.times {
  puts "Hello!"
}
```

هنا بدل مانكتب

```
puts “Hello!”
puts “Hello!”
puts “Hello!”
puts “Hello!”
puts “Hello!”
```

استخدمنا times

تقدر تستخدمها بالصورة هذه:

```
5.times do
  puts "Hello!"
end
```

بإستخدام do .. end

upto/downto

```
1.upto(4){
  puts "Hello!"
}

5.downto(0){
  puts "Hello!"
}
```

مشابهه لنفس الفكرة! من ال 1 حتى ال 4 نفذ ال Block التالى وزيد ال 1
والثانية من 5 لحد 0 نفذ ال block التالى ونقص 1

هل لاحظت إننا قلنا زيد ونقص بدون معرفة ما نزيده وما نطرحه ؟
بكل بساطة هو iterator وهو متغير يتحدد خلال loop

لاحظ التالى مثلا

```
5.times {|i|
  puts "I: " << i.to_s
}
```

الخرج:

```
I: 0
I: 1
I: 2
I: 3
I: 4
```

هل لاحظت إن ال i قيمتها كانت فى البداية 0 وفى كل مرة يتنفذ ال Block تزيد 1 ؟

تستطيع أن تكتبها كالتالى

```
5.times do|i|
  puts "I: " << i.to_s
end
```

جميل انت الآن فهمت اللعبة، دعنا نشوف ال upto وال downto ثانية:

```
0.upto(5) do |i|
  puts "I: " << i.to_s
end
```

الخرج سيكون كالتالي:

```
I: 0
I: 1
I: 2
I: 3
I: 4
I: 5
```

تستطيع أن تكتبها على الصورة التالية بإستخدام ال { }

```
0.upto(5) {|i|
  puts "I: " << i.to_s
}
```

لاحظ إن ال iterator قيمته هنا تبدأ من ال 0 وكل مرة تزيد قيمته بمقدار 1 لحد ماقيمته = 5 وينتهى تنفيذ ال Block

```
5.downto(0) { |i|
  puts "I: " << i.to_s
}
```

الخرج سيكون كالتالى

```
I: 5
I: 4
```

```
I: 3
I: 2
I: 1
I: 0
```

لاحظ إن ال iterator قيمته هنا بدأت من ال 5 لحد ال 0 وفى كل مرة ينقص قيمته بمقدار 1

تستطيع أن تستخدمها كالتالى ب do.. end

```
5.downto(0) do |i|
  puts "I: " << i.to_s
end
```

while

جميل ، أشهر حلقة تكرار فى معظم اللغات هى حلقة التكرار while

```
loopCounter=0 #set

while loopCounter<5 #condition
        puts “Loop Counter: # “ << loopCounter.to_s
        loopCounter += 1 #increment
end
```

طالما ال loopCounter اقل من 5 (الشرط)

نفذ ال Block التالى

```
        puts “Loop Counter: # “ << loopCounter.to_s
        loopCounter += 1
```

ونزيد قيمة ال counter بمقدار 1 وهكذا لحد ماتصل قيمته إلى 5 ويخرج من ال loop

الخرج:

```
LoopCounter: # 0
LoopCounter: # 1
LoopCounter: # 2
LoopCounter: # 3
LoopCounter: # 4
```

بنفس النظام نستطيع أن نعملها تنازلى

```
loopCounter=5 #set
while loopCounter>0
   puts "LoopCounter: # " << loopCounter.to_s
   loopCounter -= 1 #decrement the counter
end
```

الخرج:

```
LoopCounter: # 5
LoopCounter: # 4
LoopCounter: # 3
LoopCounter: # 2
LoopCounter: # 1
```

مثال أخير

```
loopCounter=0
while loopCounter<10
  puts "LoopCounter: # " << loopCounter.to_s
  loopCounter += 2 #increment the counter
end
```

الخرج هنا لزيادة مقدارها 2

```
LoopCounter: # 0
LoopCounter: # 2
LoopCounter: # 4
LoopCounter: # 6
LoopCounter: # 8
```

مثال:

```
loopCounter=10 #set
while loopCounter>0
   puts "LoopCounter: # " << loopCounter.to_s
   loopCounter -= 2 #decrement the counter
end
```

الخرج:

```
LoopCounter: # 10
LoopCounter: # 8
LoopCounter: # 6
LoopCounter: # 4
LoopCounter: # 2
```

for

حلقة تكرار for هى أحد أشهر حلقات التكرار فى معظم اللغات

وهنا فى Ruby هى اقرب لحلقة التكرار foreach

مثال:

```
ary=[1, 2, 3, 4, 5]
for el in ary
  puts "el: " << el.to_s
end
```

لاحظ

for .. in .. end

بالعربى لكل عنصر فى ال ary نفذ ال Block التالى

تستطيع أن تستخدمها فى while كالتالى

```
ary=[1, 2, 3, 4, 5]

index=0
while index<ary.length
  puts "el: " << ary[index].to_s
  index += 1
end
```

صف Array يقدم طريقة اخرى ل for

```
ary=[1, 2, 3, 4, 5]


ary.each{|el|
  puts "el: " << el.to_s
}
```

وهى بإستخدام iterator ايضا

الخرج:

```
el: 1
el: 2
el: 3
el: 4
el: 5
```

وتستطيع أن تستخدمها كالتالى ب do.. end

```
ary.each do |el|
  puts "el: " << el.to_s
end
```

until

until هى حلقة تكرار غير موجودة بكثرة فى معظم اللغات

```
lc=0 #set

until lc>5 #condition
  puts "lc: " << lc.to_s
  lc += 1 #increment
end
```

الخرج:

```
lc: 0
lc: 1
lc: 2
lc: 3
lc: 4
```

```
lc: 5
```

لاحظ إن حلقة التكرار معناها التالي:

حتى تصبح ال lc > من ال 5 نفذ البلوك التالى

```
  puts "lc: " << lc.to_s
  lc += 1 #increment
```

الفصل الثامن: السلاسل النصية

#جلسة السلاسل النصية

```
s="Hello, World. Sup!?"

puts s
#output: Hello, World. Sup!?
```

length

هى طريقة تعيد عدد حروف في سلسلة نصية string

```
s="this is a string"
puts s.length
#output: 16
```

upcase/upcase!

ال upcase هى طريقة تقوم بإعادة نسخة من السلسلة النصية ولكن بحروف uppercase

ال upcase! هى destructive طريقة تقوم بالتعديل على السلسلة النصية نفسها وتحويلها ل uppercase

```
x=s.upcase #returns a copy "upcase"
puts x
#output: HELLO, WORLD. SUP!?

puts s
#output: Hello, World. Sup!?

s.upcase! #destructive method!
puts s
#output: HELLO, WORLD. SUP!?
```

downcase/downcase!

ال downcase هى طريقة تقوم بإعادة نسخة من السلسلة النصية ولكن بحروف lowercase

ال downcase! هى destructive طريقة تقوم بالتعديل على السلسلة النصية نفسها وتحويل الحروف إلى lowercase

```
x=s.downcase #returns a copy "downcase"
puts x
#output: hello, world. sup!?

puts s
#output: HELLO, WORLD. SUP!?
s.downcase! #destructive method

puts s
#output: hello, world. sup!?
```

capitalize/capitalize!

capitalize هى طريقة تقوم بإعادة نسخة من السلسلة النصية ولكن اول حرف فيه يكون capital

capitalize! هى destructive طريقة تقوم بالتعديل على السلسلة النصية نفسها وتحويل اول حرف فيه إلى capital

reverse/reverse!

reverse هى طريقة تقوم بإعادة نسخة من السلسلة النصية ولكن معكوس الحروف

reverse هى طريقة تستخدم فى عكس السلسلة النصية نفسها

```
s="this is a string"
revS=s.reverse #returns a reversed copy of the string
puts revS

#output: gnirts a si siht

s.reverse! #modify the object, reverse it!
puts s
#output: gnirts a si siht
```

indexing

السلاسل النصية لها علاقة وثيقة بالمصفوفات. فى بعض اللغات لا تعترف بشي اسمه سلسلة نصية وتعتبرها بدلا عن ذلك مجرد مصفوفة من الحروف مثل ال C !

ففى بعض الأشياء المشتركة بين السلاسل النصية والمصفوفات مثل الفهارس ، وتبدأ أيضا بالصفر.

فإنت تستطيع أن تحصل على الحرف الموجود ب index معين

String[index]

اول حرف سيكون فهرسه هو 0

لاحظ أن القيمة التي ستعاد هي قيمة ال char من جدول ال ASCII فلذا سنستخدم chr method لتحويل ال

character إلى integer value

الحرف الأخير سيكون فهرسه هو 1-

```
puts s[0]#first, returns the ascii value
#output: 116
puts s[0].chr #converted to a char
#output: t
puts s[-1] #last, returns the ascii value
#output: 103
puts s[-1].chr #converted to a char
#output: g
```

String[From..To]

تقوم بإعادة كل الحروف من ال From لل To

```
puts s[1..4]
#output: his
```

index

هي طريقة تعيد لنا ترتيب الحروف فى string أو ترتيب اول ظهور لتعبير مثلا التالى

```
s="this is a string"
puts s.index("i")
#output 1
```

هنا الطريقة تعيد لنا ترتيب اول i هى قابلتها وهى فى ال 2 index

نقدر نعدل كيفية الإستخدام بإننا نضيف ال offset او مكان البداية فى البحث بال string كالتالى

```
s="this is a string"
puts s[8..s.length]
#a string

puts s.index("i", 8)
#output: 13
```

count

هى طريقة تقوم بإعادة عدد مرات تكرار char او expression فى ال string مثلا كالتالى

```
s="this is a string"
puts s.count("i") #count how many iS in s
#output: 3
```

empty?

هى طريقة تختبر إذا كان ال string فارغ أم لا

```
puts "it's empty" if "hello".empty?
```

include?(char)/include?(substring)

نختبر بها وجود char او substring فى ال string نفسه أم لا

```
puts s.include?("w")  #has "w" char
#output: true
puts s.include?("he") #has "he" substring ?
#output: true
```

chomp/chomp!

chop/chop!

chop: هى طريقة تقوم بحذف الحرف الأخير من ال string وإذا واجهت \r\n بتقوم بحذفهم هما الإثنين!
من غير المحبذ إستخدامها..

chop! هى نفس عمل السابقة ولكن تقوم بالتعديل على ال string لأنها طريقة destructive

إرشاد: يفضل إستخدام chomp

chomp/chomp!(separator)

chomp هى طريقة تقوم بإعادة نسخة من السلسلة النصية مع حذف الحرف الأخير separator وهو \n او \r او
كلاهما افتراضيا إلا إذا حددت إنت ال separator
chomp! هى نفس عمل السابقة وإنما التعديل بيكون على ال string نفسه لأنها destructive ميثود

```
puts "Hello, World\n".chomp
#output: Hello, World

puts "Hello, World\r\n".chomp
#output: Hello, World
puts "Hello, World\r".chomp
#output: Hello, World
puts "Hello, \nWorld".chomp
#output:Hello,
#World

puts "Hello, World.".chomp("d.")
#output: Hello, Worl
```

```
puts "Hello, World!".chomp("!")
#output: Hello, World
```

concat

هى طريقة تستخدم فى دمج اكثر من string

```
s="Hello, "
s.concat("World!")

puts s
#output: Hello, World!

s="Hello, "
s << "World!"
puts s
#output: Hello, World!

s="Hello, "
s += "World!"
puts s
#output: Hello, World!
```

Comparing

```
s1="Hello"
s2="Hello"
s3="hello"
puts "s1 equals s2" if s1.eql?(s2)
puts s1.casecmp(s3) #it doesn't care about the case
puts "s1 == s3" if s1==s3 #it doesn't match it!

#output:
s1 equals s2
0
```

scan(pattern)

هى طريقة تستخدم فى عمل مسح على نمط معين فى السلسلة النصية وتعيدهم فى صورة مصفوفة

```
chars=s.scan(/./)
p chars #inspect
#output: ["h", "e", "l", "l", "o", ",", " ", "w", "o", "r", "l", "d", ".", " ",
"s", "u", "p", "!", "?"]

s="Hello, World, Test, what?"
scann=s.scan(/\w+/) #scan for word+char
p scann
#output: ["Hello", "World", "Test", "what"]
```

split(separator)

هى طريقة تستخدم فى عمل split ل separator وتعيد array ..

```
words=s.split(' ') #split the space separator.
p words
#output: ["hello,", "world.", "sup!?"]
```

join(separator)

هى طريقة تستخدم فى عمل join لعناصر ال array وإعادة string

```
joinedString=words.join(' ')
#join the array elements with a separator
puts joinedString
#output: hello, world. sup!?

joinedString=words.join('<>')
#join the array elements with a separator
puts joinedString
#output: hello,<>world.<>sup!?
```

gsub(pat, replacement), gsub!(pat, replacement)

هى طريقة تستخدم فى إستبدال pat ب replacement

```
s="Hello"
puts s.gsub('e', '3')
#output: H3llo

puts s.gsub('lo', '10')
#output: Hel10

s="I LOVE Ruby!"
puts s.gsub(/Ruby/, "Python")
#output: I LOVE Python!
```

freeze, frozen?

Freeze هي طريقة تمنع الكائن من التعديل عليه

frozen? هى طريقة تستخدم فى إختبار هل الكائن تم تنفيذ freeze عليه أم لا

```
s="Hello"
s.freeze
```

```
s.
s << ", World!" #can't be done as "s" is frozen!
puts "freezed!" if s.frozen?
```

to_s

هى طريقة تعبر عن ال object فى حالة إستخدام puts معاه

to_i

هى طريقة تعبر عن ال object فى حالة محاولة تحويله ل int

to_f

هى طريقة تعبر عن ال object فى حالة تحويله ل float

to_a

هى طريقة تعبر عن ال object فى حالة تحويله ل array

to_hash

هى طريقة تعبر عن ال object فى حالة تحويله ل hash

مثال على ال OOP/Strings بإضافة طريقة جديدة لل String Class وهى atbash

```
class String
  def atbash
    a=('a'..'z').to_a
    aRev=a.reverse
    s=self.downcase
    res=""
    for char in s.scan(/./)

      if a.include?(char)
        res << aRev[a.index(char)]
      else
        res << char
      end

    end
    return res
  end
end

s="Hello"
puts s.atbash
#output: svool
```

لاحظ إن ال self هى ال string !

مثال آخر وهو إضافة ال rot13 method لل String Class

```
class String

  def rot13
    self.tr("A-Ma-mN-Zn-z","N-Zn-zA-Ma-m")
  end

end

puts “ahmed”.rot13
#output: nuzrq
```

Heredoc string

بكل بساطة هو string يمتد على اكثر من سطر multi-line string

```
heredocString=<<start
Hello,
i
am
a multi-line
string
start
puts heredocString
```

لاحظ إنك أنت الذي تحدد ال tag الخاص بالبداية بس بشرط إنك تقفل بها ال string وهنا انا إستخدمت start مثلا!

```
#output:
Hello,
i
am
a multi-line
string
```

ملحوظة بخصوص ال symbols

اعتبر ال symbol كانه string ولن يتغير خلال ال runtime ولن تحتاج فيه للطرق الخاصة بال string

للمزيد اطلع هنا

http://www.troubleshooters.com/codecorn/ruby/symbols.htm

الفصل التاسع: IO (الدخل / الخرج)

ذكرنا سابقا ان puts/print/gets أنها عبارة عن دوال، أليس كذلك ؟ الإجابة خطأ بالطبع لأن Ruby is Fully OOP ف puts/print/gets يقعوا تحت ال Kernel module وهى module مثلها مثل أي module ولكن الذي يميزها إنها implemented by default

puts
تستخدم فى عملية الطباعة على ال stdout

print
تستخدم فى عملية الطباعة على ال stdout
gets

تستخدم فى قراءة string من ال stdin

ruby مثلها مثل العديد من اللغات بتقدم classes جاهزة للتعامل مع المجلدات والملفات

المجلدات

للتعامل مع المجلدات نستخدم ال Dir Class

العمل الحالي

pwd, getwd

هما طريقتين يستخدما فى الحصولعلى ال Current Path/Current Working Dir

تغيير المجلد

chdir(to)

إنشاء المجلدات

mkdir(new_dir_name, permissions)

نستخدم الطريقة mkdir فى إنشاء directory جديد بحيث إننا نحدد الإسم فى ال new_dir_name والتصاريح فى ال permissions

حذف المجلدات

```
Dir.delete(dir_name)
Dir.unlink(dir_name)
Dir.rmdir(dir_name)
```

نستخدم الطرق التالية وهى delete, unlink, rmdir

تصفح مجلد

entries

هى طريقة تعيد مصفوفة فيها كل محتويات المجلد، وعند عمل حلقة تكرار بسيطة على مصفوفة المدخلات، سيظهر لدينا كل محتويات المصفوفة:

```
Dir.chdir('C:\ruby\bin\\') #change to the dir

for entry in Dir.entries(Dir.getwd)
  puts entry
end

Dir.entries(Dir.getwd) {|entry|
  puts entry
}

Dir.entries(Dir.getwd) do |entry|
  puts entry
end


Dir.foreach(Dir.getwd) {|entry|
  puts entry
}

Dir.foreach(Dir.getwd) do |entry|
  puts entry
```

```
end
```

كائنات المجلدات

تقدر تعمل Dir Stream او Dir Object او Dir Handler على حسب ما تريد، تفرض التسمية بإستخدام طريقة open او new من ال Dir Class

اولا دعنا نتفق على شئ مهم .. الذي يفتح شيء يجب أن يقفله ... مثلا أنت شغلت الحاسوب لتنجز مهمة معينة، فبعد إنجازها تقفل الجهاز.

نفس النظام فى التعامل مع ال Dirs/Files

إذا فتحت مجلد يجب أن تقفله .. واذا فتحت File يجب أن تقفله أيضا

جميل جدا... نفتح ال Stream كالتالي

```
dirObj=Dir.open('C:\ruby\bin\\')

dirObj.close
```

تستطيع أن تستخدم الطريقة new مثلما اشرت سابقا ..

path

للحصول على ال path

```
puts dirObj.path
#output: C:\ruby\bin\
```

tell

تعيد ال current position فى ال Directory

read

تقرأ ال next entry

rewind

تعمل set لل current position فى ال Directory إلى البداية - 0 -

each

نستخدمها فى محاولة ال iteration على ال entries

```
dirObj.each do |entry|
     puts entry
end

dirObj.each {|entry|
     puts entry
}
```

مثال:

```
dirObj=Dir.open('C:\ruby\bin\\') #open
#dirObj=Dir.new('C:\ruby\bin\\')
puts dirObj.path

puts dirObj.tell #the current position
puts dirObj.read #what's it ?

puts dirObj.tell #the current position
puts dirObj.read #what's it ?

puts dirObj.tell #the current position
puts dirObj.read #what's it ?


#set the current position to 0
dirObj.rewind
puts dirObj.tell
dirObj.close

#output
C:\ruby\bin\
0
.
1
..
2
erb.bat
0
```

الملفات

اولا دعنا نتفق على شئ مهم:

```
r    => read only
r+  => read/write
w   => write only
w+ => write/read
a    => append at the end or create
a+  => append at the start or create
b    => binary
```

delete(file_path)

هى طريقة تستخدم لحذف الملف

او تستطيع أن تستخدمها كالتالى

```
delete(file1, file2, file3)
```

rename(from, to)

هى طريقة تستخدم فى عمل rename لل file

ctime(file_path) (creation time)

هى طريقة تعيد تاريخ انشاء ال file

mtime(file_path) (last modification time)

هى طريقة تعيد تاريخ اخر تعديل على ال file

atime (file_path) (last access time)

هى طريقة تعيد تاريخ اخر مرة تم إستخدام ال file

```
puts File.mtime("C:\\ls.rb")
puts File.ctime("C:\\ls.rb")
puts File.atime("C:\\ls.rb")

#output:
Mon Nov 26 08:35:53 EET 2007
Mon Nov 26 08:35:53 EET 2007
Mon Nov 26 08:35:53 EET 2007
```

zero?

تختبر إذا كان طول الملف يساوي صفر أم لا.

اقرب مثال لها هى ال files اللى تنشيء باستخدام touch

exist?(file_name)

هى طريقة تستخدم فى إختبار هل ال file موجود أم لا

exists?(file_name)

نفس سابقتها ولكن obsolete

readable?(file_name)

هل الملف قابل للقرآءة

writeable?(file_name)

هل الملف قابل للكتابة

Executable?(file_name)

هل الملفقابل للتنفيذ

size(file_name), size?(file_name)

المساحة

ftype

تستخدم فى تحديد نوع الملف.

directory?(file_name)

تستخدم فى إختبار مجلد هل مطابق للاسم المعطى أم لا.

مثلما اتفقنا أي استخدام لمصدر يجب أن يقفل في نهاية البرنامج

الكتابة

syswrite, puts, putc

syswrite و puts يستخدموا فى كتابة string فى ال file

putc بتستخدم فى كتابة character

```
#Create a file Object in Write mode, if it exists it'll be overwritten
fileObj=File.new('C:\tstRuby.txt', 'w')

#Print a string to the file by using syswrite method of the file Object
fileObj.syswrite("Hello from Ruby IO!")

#Close the file object!
fileObj.close


Reading

fileObj=File.new('C:\tstRuby.txt', 'r')

#Read it
fileSrc=fileObj.gets

#print it
puts fileSrc

fileObj.close() #Close it.
```

each_byte

هى طريقة تستخدم فى عمل iteration على كل char

```
#Create a file Object in read mode
fileObj=File.new('C:\tstRuby.txt', 'r')

#Read it
fileObj.each_byte {|char|
  putc char
}

#Close it
fileObj.close()


FileCopy.rb
```

الفصل العاشر: معالجة الأستثناءات

عملية التنقيح Debugging هي عملية وضع البرنامج تحت الإختبار قبل أن ينزل السوق .. ولو حصل في البرنامج خطأ ما.. فإنه سيظهر مباشرة للمستخدم والمستخدم سيرسل لفريق التطوير بنوع الخطأ الحاصل، وهم بدورهم سيعالجوا الخطأ في شفرة المصدرية للبرنامج وبعدها يقوموا باختبار البرنامج مرة أخرى و إنزاله في السوق وهكذا .. توجد تقنية أخرى منتشرة تسمى معالجة الاستثناءات Exceptions بحيث تقوم بمعالجة الخطأ وإظهار رسالة واضحة للمستخدم بطريقة مفهومة .. مثلا عملية القسمة على صفر غير منطقية بالمرة أليس كذلك ؟

جرب أن تنفذ الكود التالي:

```
puts 1/0
```

ستجد التالي:

```
divided by 0 (ZeroDivisionError)
```

فكيف سنتحكم بهذه العملية؟

الحل هو بإستخدام begin/rescue/end

الصورة العامة

```
begin
           #your normal code goes here.
           #error!

rescue
          #rescue code

ensure
         #always happen!
end

begin
  puts 1/0
rescue ZeroDivisionError
  puts "Never studied math, you genius? You are dividing by zero!"
  exit(1) #terminate
end
```

سنقوم بمعالجة المشكلة فى rescue او (catch/except block) بإن نظهر رسالة التالية:

```
 "Never studied math, you genius? You are dividing by zero!"
```

ونخرج من البرنامج بإستخدام exit

جميل ، الآن أتى وقت ensure (ال finally)

ensure كتلها يجب أن تنفذ!

```
begin
  puts 1/0
rescue ZeroDivisionError
  puts "Never studied math, you genius? You are dividing by zero!"
  exit(1) #terminate
ensure
  puts "This is always appeared!"
end
```

إذا سأل شخص ما وجه الإستفادة من ensure ؟

ensure تستطيع أن تستخدمها فى قفل مصدر المفتوح مثلا ملف او اتصال وهكذا

جميل، تستطيع أن تفصل Exceptions على مزاجك :D

مثال تطبيقى:

على فرض أن الطفل سمع دق الجرس ، وهو لا يعرف من القادم، فكيف سيتعامل مع الموقف؟

سنجعله يظهر معرفه id وإلا سيقفل الباب و يطلب الشرطة، أليس كذلك؟

الأول سنعمل Error مخصوص ليعبر عن مشكلتنا وهى أن شخصا غريبا دق الجرس

```
class IntruderKnocksError < StandardError

end
```

بعد هذا ، سنضع كودنا:

```
begin
      door_closed=false

      who="intruder"
      if who=="intruder"
             raise IntruderKnocksError, "Can't open for intruders."
      end
rescue IntruderKnocksError => e
  puts e.message #Error message
  p     e.backtrace #backtrace! ( Where was the exception raised )
  puts "show me some id.. [y/n]: "
  ans=gets.chomp!.downcase
  if ans=='y'
    who="known"
    puts "hi, i know you!"
  else
    puts "you wanna kidnap me? calling police.."
    exit 1
  end

ensure
```

```
  door_closed=true
end
```

طبعا الكلام هدا يختلف جدا عن الواقع لكنه نفس الفكرة

علاوة

Reflection

بكل بساطة هى القدرة على التعامل مع ال meta data بحيث تستطيع أن تعرف ماذا في المصدر و أي صف أوي طريقة يحتويها، و أي بارميتر تأخذها كل طريقة وهكذا...

```
s="Hola!"
```

هي مجرد كائن object عادى

class

هى طريقة تستخدم فى الحصول على نوع ال class الذى تم عمل ال object منه

```
#s is an object from what?
puts s.class
#output: String
```

class.name

تستخدم فى الحصول على إسم ال class

```
#what is the name of this class ?
puts s.class.name
#output: String
```

superclass

تستخدم فى الحصول على ال parent class لل class

```
#from what String  was inherited?
puts String.superclass #object is the parent class of 'em all!
```

included_modules

للحصول على ال modules التي عمل لها احتوى include فى ال class

```
#the included modules in String
p String.included_modules
#output: [Enumerable, Comparable, Kernel]
```

```
p Object.included_modules
#output: [Kernel]
```

object_id

كل object ليه unique id وهذه الطريقة تستخدم فى الحصول على ذلك ال id

```
#each object has a unique id.
s1="Hi"
puts s1.object_id  #6

s2="Hi"
puts s2.object_id  #8
```

لاحظ التالى فى حال إن كائنين يشيروا لنفس ال object

```
s1="Hi"
s2=s1 #points to s1

puts s1.object_id #6
puts s2.object_id #6
```

constants

هى طريقة تعيد array فيها كل ال constants الموجودة

```
p Math.constants
#output: ["E", "PI"]
```

local_variables

تستخدم فى الحصول على ال local variables فى البرنامج

```
#get the local variables
p local_variables
#output: ["s", "s1", "s2"]
```

global_variables

تستخدم فى الحصول على ال global variables

```
$globString="What's up????"
$anotherGlobString="CRUEL, WORLD!"

#gets the global variables
p global_variables
#output
#["$FILENAME", "$LOADED_FEATURES", "$anotherGlobString",
#  "$VERBOSE", "$globString", "$PROGRAM_NAME", "$LOAD_PATH",
#  "$DEBUG", "$stdin", "$KCODE", "$stderr", "$stdout", "$defout",
#  "$deferr", "$-I", "$`", "$-K", "$\\", "$_", "$!",
```

```
#  "$\"", "$-a", "$-d", "$~", "$@", "$?", "$>",
#  "$=", "$<", "$:", "$0", "$.", "$/", "$,", "$-n",
#  "$*", "$SAFE", "$+", "$-p", "$&", "$'", "$$", "$-l"]
```

instance_variables

بتستخدم للحصول على ال instance_variables

```
p instance_variables # []
```

methods

تستخدم للحصول على ال methods الخاصة بال class

```
#get all of the methods in string
methodsAry=String.methods
p methodsAry
#output:
#["new", "superclass", "allocate", "inherited", "initialize_copy",
.....................
#  "class_eval", ">", "<", "private_class_method",
,
#  "protected_methods", "nil?", "freeze", "is_a?", "eql?"]
```

instance_methods

تستخدم فى الحصول على ال methods الخاصة بال instance

```
#instance methods
p String.instance_methods
```

private_methods

تستخدم فى الحصول على ال private methods الموجودة!

```
#get the private methods.
p String.private_methods
```

public_methods

للحصول على ال public methods

protected_methods

للحصول على ال protected methods

singleton_methods

يستخدم فى الحصول على ال singleton methods

protected_methods

للحصول على ال protected methods

ونفس الشئ مع

[private|protected|public]_instance_methods

respond_to?(:methodName)

تستخدم فى معرفة هل يمكن إستخدام الطريقة methodName مع ال object ام لا

```
s="Hello"
puts s.respond_to?(:upcase) #true: can use upcase method
puts s.respond_to?(:keys) #false: can't use keys method
```

class hierarchy

هل تساءلت كيف تحدد بيئة التطوير IDEs ال super classes الخاصة بصف معين مشتق منهم ؟ تابع المثال التالى ولن تحتار مرة أخرى !!

```
class First < String

end

class Second < First

end

class Third < Second

end

c=Third

while c
  print(c)
  print(" < ")
  c=c.superclass

end
puts

#output: Third < Second < First < String < Object
```

الفصل الحادى عشر:XML & Ruby

هى اختصار ل EXtensible Markup Language وهى واضحة بأنها Markup language مثل ال HTML ولكن يوجد فرق !!

XML vs HTML

الإثنان يحتويان على وسوم tags لكن الفرق أن ال HTML تستخدم tags محددة وجاهزة ولكن ال XML انت الذي تخترع فيها ال tags الخاصة بك.
فرق مهم جدا أيضا وهو أن HTML للعرض لكن ال XML لتخزين البيانات، بكل تأكيد تستطيع أن تستخدمها فى العرض ولكن ليس هذا هدفنا من هذا الموضوع.
هناك تقنيات كثير تعتمد على ال XML ، حتى في عمل الواجهات الرسومية GUI! حيث تحتفظ المعلومات في ملف ب xml syntax .. لن أكون مبالغا اذا قلت ان ال NET. وال Java قائمين على ال xml فى أشياء كثيرة.

على فرض اننا نملك ملف كالتالى books.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : books.xml
    Created on : April 19, 2008, 10:01 AM
    Author     : ahmed
    Description:
        Purpose of the document follows.
-->

<books>
       <book>
            <id>1</id>
            <name>Introduction to Ruby</name>
            <author>Ahmed Youssef</author>
            <price>20</price>
       </book>

       <book>
            <id>2</id>
            <name>Introduction to Python</name>
            <author>Ahmed Youssef</author>
            <price>40</price>
```

```
        </book>

        <book>
             <id>3</id>
             <name>Introduction to C </name>
             <author>Ahmed Mostafa</author>
             <price>70</price>
        </book>

       <book>
             <id>4</id>
             <name>Introduction to Perl</name>
             <author>M_SpAwN</author>
             <price>40</price>
        </book>

</books>
```

واضح أنه منظم ومقسم ل books وتضم book element وكل واحد فيهم يضم id و name و author و price
لاحظ اى وسم بهذه الصورة <start></start> يسمى عنصر element
ال XML فى ال configuration files تكون مريحة مثل هذه، بعكس النص المجرد فى .ini مثلا او حتى في ملفات اعداد grub و lilo

العملية كلها تتم على عدة خطوات:

1- أنك تعمل import ل REXML كالتالى

```
#STEP 1 (import rexml module)

require "rexml/document"
include REXML
```

2- انك تدخل على ملف ال xml كالتالي، مثلا عن طريق انشاء Document object من ال Document class الموجود فى ال unit التى حملنها:

```
#STEP 2 (load the xml document)
xmlDOC=Document.new(File.new("books.xml"))
```

تستطيع أن تستخدم ال HEREDOC string ولكن لا أحب موضوع ال Hard Coding فى السكربت او البرنامج نفسه

ما رأيك بأن نطبع اسم كل كتاب في ملف xml ؟

```
xmlDOC.elements.each("books/book/name") do |element|
  puts element.text
```

```
end

#Output
                Introduction to Ruby
                Introduction to Python
                Introduction to C
                Introduction to Perl
```

لكل عنصر بإسم name ستتم طباعة ال text (لاحظ ان element هو صف مستقل بذاته ونحن لا نريد غير text)

اذا كتبت puts element سيظهر لك شئ كالتالي:

```
<name>Introduction to Ruby</name>
<name>Introduction to Python</name>
<name>Introduction to C </name>
<name>Introduction to Perl</name>
```

وبنفس الفكرة إذا أردنا أسماء المؤلفين، سنقوم باستبدال name ب author

لمحة سريعة:

```
authors=[]
xmlDOC.elements.each("books/book/author") do |element|
  authors.push(element.text)
end

#uniq! it
authors.uniq!
p authors

#output ["Ahmed Youssef", "Ahmed Mostafa", "M_SpAwN"]
```

آخر لمحة وهى أننا نريد معرفة سعر الكتب الكلي

```
#get sum of prices ?
sum=0

xmlDOC.elements.each("books/book/price") do |element|

  sum += element.text.to_i
end

puts "Total: "+ sum.to_s

#output: 170
```

الاسلوب الحالي يسمى أسلوب DOM وهو يعتمد على tree (حيث يخزن كل الملف فى tree structure فى الذاكرة)

يوجد أسلوب آخر، شخصيا أفضله و يوافقنا الرأي كثيرون، وهو أسلوب SAX وهو يعتمد على ال events انه يبدأ فى tag معين (فهو يتعامل مع ال tag وال attributes)

```
<start attr1=val attr2=val2> DATA </start>
```

ويدير البيانات

```
<start attr1=val attr2=val2> DATA </start>
```

وانه ينهى tag معين

```
<start attr1=val attr2=val2> DATA </start>
```

دعنا نجرب نفس المثال الخاص بالحصول على الثمن الكلى للكتب من ملف books.xml

1- اعمل load لل rexml كالتالى

```
  require "rexml/document"
  require "rexml/streamlistener"
  include REXML
```

2- أنشئ ال ContentHandler او ال Streamer (بيثون مأثرة شوية:])

سنعرف ال callbacks مثلا اذا بدأ فى tag او بدأ فى البيانات الخاصة بال tag او ينهى ال tag كالتالى:

```
  class BooksStreamer
    include StreamListener

    def initialize
      @inPrice=false
      @sum=0
    end

    def tag_start(tag_name, attrs)
      puts “Starting #{tag_name}”
      if (tag_name=="price") then
        @inPrice=true
      end
    end

    def tag_end(tag_name)
      #puts "Ending #{tag_name}"
      @inPrice=false
    end

    def text(data)
        if @inPrice then
            @sum += data.to_i
        end
    end


    def get_total_sum

      return @sum
```

```
    end
  end
```

اولا tag_start هي اول callback تستدعى لما ال parser يبدأ في tag معين ولاحظ ان ال parser سيمر على ال tag_name وخصائصه

```
<start attr1=val attr2=val2 ....> DATA </start>
    def tag_start(tag_name, attrs)
      if (tag_name=="price") then
      end
    end
```

جيد، نحن الان لن نهتم بغير price tag فإذا دخل ال parser في tag بإسم price فإننا نريد أن نوضح هذه المعلومة للكائن وهي أننا نستخدم instance variable يشير إلى اننا في ال price tag كالتالي

```
   @inPrice=true
```

فتتحول إلى:

```
    def tag_start(tag_name, attrs)
      #puts "Starting #{tag_name}"
      if (tag_name=="price") then
        @inPrice=true
      end

    end
```

جميل جدا.. وبالمنطق إذا مر المفسر على وسم </price> ؟

```
<start attr1=val attr2=val2> DATA </start>
```

فبكل تأكيد هو لن يكون في price tag فنعمل reset للمتغير الذي يشير هل ال parser في price tag او لا كالتالي:

```
  def tag_end(tag_name)
      #puts "Ending #{tag_name}"
      @inPrice=false
    end
```

نأتي لأبسط شئ وهو إلى الحصول على المجموع

1- عرّف instance variable بإسم sum@ وأعطيه قيمة = 0

```
    def initialize
      @inPrice=false
      @sum=0
    end
```

2- فى جزئية ال data حولها إلى integer وأضفها على ال sum@

```
<start attr1=val attr2=val2> DATA </start>
```

كالتالى مثلا

```
    def text(data)
        if @inPrice then
            @sum += data.to_i
        end
    end
```

للحصول على ال sum اعمل getter كالتالى

```
    def get_total_sum

      return @sum

    end
```

جميل جدا، يبقى كيف نستخدم صفنا هذا؟

1- اعمل نسخة من ال BooksStreamer كالتالى:

```
bs=BooksStreamer.new
```

2- مرر ملف المصدر و كائن BooksStreamer إلى Document.parse_stream كالتالي:

```
 Document.parse_stream(File.new("books.xml"), bs)
```

هكذا يكون bs جاهزا حسب تطبيقاتك فى start, end, text فكل ما عليك هو انك تستدعى get_total_sum كالتالى:

```
 puts bs.get_total_sum

#output: 170
```

الفصل الثاني عشر: Ruby Gems

اذا كنت قد تعاملت مع منصة لينكس فقد استخدمت أداة مشابهه لـ apt-get لتحميل البرامج/المكتبات. بنفس النظام ال gem هو تطبيق أو مكتبة عمل كحزمة و لكن ليس مثل.deb او .rpm. لكن .gem على Ruby Platform جميل جدا

المميزات

1- امكانية تثبيت/حذف ال Gem Packages
2- إلغاء مشكلة الاعتماديات
3- سهولة التحكم في الحزم
4- الاستعلام/البحث وعرض الحزم المتوفرة
5- سهولة إنشاء الحزم
6- سهولة الإطلاع على الوثائق الخاصة بال gemS

التثبيت

1- قم بالتحميل من الوصلة التالية http://rubyforge.org/frs/?group_id=126
2- شغل ملف setup.rb كالتالى

```
ruby setup.rb
```

اكتب

```
gem -v
```

او

```
gem –version
```

```
#output:
0.9.4
```

بهذا ثبت البرنامج.

اعرض ال environment

gem environment

كالتالى

```
C:\Documents and Settings\ahmed>gem environment
RubyGems Environment:
```

```
  - VERSION: 0.9.4 (0.9.4)
  - INSTALLATION DIRECTORY: c:/ruby/lib/ruby/gems/1.8
  - GEM PATH:
     - c:/ruby/lib/ruby/gems/1.8
  - REMOTE SOURCES:
     - http://gems.rubyforge.org
```

اهم الخيارات

1- install

للقيام بتثبيت gem معين سواء local او remote

```
gem install rake
gem install rails
```

2- list

لعرض ال gems

```
gem list F
```

عرض ال gems اللى بتبدأ بحرف ال F

3- uninstall

لحذف gem معينة

```
gem uninstall rails
```

4- search

للبحث عن gem ما

```
search <gem> --local
```

استخدام –local فى حال البحث local

```
search <gem> --remote
```

استخدم –remote للبحث remote

```
search <gem> --both
```

استخدم –both للبحث local و remote

help -5

للحصول على المساعدة

للحصول على المساعدة بخصوص command معين

```
gem help <command>
```

لعرض كل ال commands

```
gem help commands
```

لعرض امثلة على الإستخدام استخدام

```
gem help examples
```

للتعامل مع الوثائق الخاصة بال gems شغل ال gem_server كالتالى

```
gem_server
```

URLShorter

توجد مواقع كثيرة تقدم خدمة تصغير ال url مثل tinyurl على سبيل المثال!

1- قم بتثبيت gem shorturl كالتالى

```
C:\Documents and Settings\ahmed>gem install shorturl
Successfully installed shorturl-0.8.4
Installing ri documentation for shorturl-0.8.4...
Installing RDoc documentation for shorturl-0.8.4...
```

2- اعمل import كالتالى

```
require 'shorturl'
```

3- اختيار السرفس اى موقع اللى عايز تستخدمه لتصغير ال urls ؟

اكتب puts ShortURL.valid_services سيعرض لك قائمة بكل المتاح مثلا

```
puts ShortURL.valid_services
```

مثلا

```
lns
shortify
makeashorterlink
fyad
rubyurl
orz
skinnylink
d62
shorl
tinyurl
moourl
```

4- استدعى Shorten method كالتالى

```
ShortURL.shorten(link,service)
```

5- نكتب الكود الخاص بنا بحيث يكون قابل للاستخدام اكثر من مرة فى class مثلا او Function كالتالى

```
require 'shorturl'

require 'shorturl'

def shortize(url)
  return ShortURL.shorten(url, :tinyurl)
end

puts shortize("www.linuxac.org/forum")

#output: http://tinyurl.com/5rrela
```

طبعا ستتضح الميزة مع ال URLs الطويلة جدا.

FasterCSV

ال CSV هى اختصار ل Comma Separated Values مثلا بينات مخزنة كالتالى

```
name, age, sex
```

مثل

```
ahmed, 18, male
tina, 18, female
omar, 18, male
ayman, 17, male
rogina, 20, female
```

1- ثبت ال fastercsv gem كالتالى

```
C:\Documents and Settings\ahmed>gem install fastercsv
Successfully installed fastercsv-1.2.3
Installing ri documentation for fastercsv-1.2.3...
Installing RDoc documentation for fastercsv-1.2.3...
```

2- اعمل import ل fastercsv كالتالى

```
require 'fastercsv'
```

3- حدد البيانات سواء فى نفس الملف او خارجه ، سأستخدم ال HEREDOC string هذه المرة.

```
data =<<d
ahmed, 18, male
tina, 18, female
omar, 18, male
ayman, 17, male
rogina, 20, female
d
```

استدعى ال parse method كالتالى

```
FasterCSV.parse(data) do |rec|
  name=rec[0]
  age =rec[1]
  sex =rec[2]

  puts "Name: #{name}, Sex: #{sex}, Age:#{age}"

end
```

```
#Output:
Name: ahmed, Sex:  male, Age: 18
Name: tina, Sex:  female, Age: 18
Name: omar, Sex:  male, Age: 18
Name: ayman, Sex:  male, Age: 17
Name: rogina, Sex:  female, Age: 20
```

المصادر :

1- http://rubygems.org/
2- Practical Ruby Gems

الفصل الثالث عشر: قواعد البيانات (SQLite/ActiveRecords)

المتطلبات : خلفية بسيطة فى SQL

الآن SQLite أصبحت جزء شبه أساسي من مكتبات معظم اللغات ، وتوجد بشكل افتراضي في عدة بيئات تطويرية ، مثل rails حاليا

ما الذي يميز SQLite ؟

1- الحجم الصغير
2- لا تحتاج إلى خادم فهى Serverless Database Engine
3- لا تحتاج لإعدادات Zero-Configurations

SQLite مستخدمة فى عدد كبير جدا من التطبيقات ك backend Databse
http://www.sqlite.org/mostdeployed.html

انا أفضل استخدم SQLite ويليها MySQL او Postgres وهذا ايضا يعتمد على حسب التطبيق ولكن SQLite will just cut it!

1- حمل

```
http://www.sqlite.org/download.html
```

2- ثبت ال gem كالتالى

```
gem install sqlite3-ruby
```

طبعا بالصلاحيات المناسبة

3- اعمل import ل sqlite3 كالتالى

```
require 'sqlite3'
```

4- تصميم ال Class سيكون كالتالى

```
require 'sqlite3'

class PhonerLite

  def initialize(dbname)
    @dbname=dbname
    @db=SQLite3::Database.new(dbname)
    @db.results_as_hash=true #row['username']
    create_table
```

```
  end

  def create_table
    #Create table.
    begin

      sqlstmt="CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT,
username VARCHAR(55) UNIQUE, phonenum VARCHAR(55))"
      @db.execute(sqlstmt)
      #puts "Table Created.."

    rescue Exception => e
      #puts "Table exists."

      puts e.message
      #e.backtrace

    end

  end

  def display_user(user)
    puts "-----------------------"
    id=user['id'].to_s
    puts "ID: #{id}"
    puts "Name: #{user['username']}"
    puts "Phone: #{user['phonenum']}"
    puts "-----------------------"
  end

  def find_user(username)
    begin
      @db.execute("SELECT * FROM users WHERE username LIKE '#{username}%'") do
|user|
       display_user(user)
      end
    rescue Exception => e
      puts "Not found!"
    end
  end

  def delete_user(username)
    begin
      delstmt="DELETE FROM users WHERE username='#{username}'"
      @db.execute(delstmt)
      puts "#{username} is deleted."
    rescue Exception => e
      puts e.message

    end
  end

  def add_user(username, phonenum)
    begin
      @db.execute("INSERT INTO users (username, phonenum) VALUES(?,?)",
```

```
username, phonenum)
      puts "#{username} is added!"
    rescue Exception => e
      puts e.message
    end
  end



  def select_all
    selectstmt="SELECT * FROM users"
    begin
      @db.execute(selectstmt) do |rec|
        display_user(rec)
      end
    rescue Exception => e
      puts e.message

    end
  end

  def delete_all
    delstmt="DROP TABLE users"
    @db.execute(delstmt)
    #create new one
    puts "Table Dropped!"
    create_table
  end

  def update_user(username, newphone)
    updatestmt="UPDATE users SET phonenum='#{newphone}' WHERE
username='#{username}'"
    begin
      @db.execute(updatestmt)
      puts "#{username} is updated!"
    rescue Exception => e
      puts e.message
    end
  end

end
```

للحصول على اتصال مع قاعدة بيانات SQLite نستخدم التالى

```
SQLite3::Database.new(DBNAME)
```

مثل مافعلنا فى ال Constructor كالتالى

```
  def initialize(dbname)
    @dbname=dbname
    @db=SQLite3::Database.new(dbname)
    @db.results_as_hash=true #row['username']
    create_table

  end
```

بننشئ instance variable بإسم db من ال SQLite3::Database class ونمرر له اسم قاعدة البيانات المطلوبة فى

حال وجودها سيتم عمل اتصال وفى حال عدم وجودها سيتم إنشاءها

استخدام results_as_hash لإمكانية الوصول لل data كأنها فى hash مثلا بإستخدام اسم ال Column

```
record['username']
```

بمجرد ما يتم الإتصال سنحاول ننشئ جدول نخزن فيه البينات باستخدام ال create_table method المعرفة كالتالى

```
  def create_table
    #Create table.
    begin

      sqlstmt="CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT,
username VARCHAR(55) UNIQUE, phonenum VARCHAR(55))"
      @db.execute(sqlstmt)
      #puts "Table Created.."

    rescue Exception => e
      puts "Table exists."

      puts e.message
      #e.backtrace

    end

  end
```

ال Database Connection Object اللى هو db@ ليه امكانية تنفيذ أوامر على الجدول باستخدام ال execute method

هذا الجدول يتكون من 3 columns

1- هو ال id وهو PrimaryKey و بيزداد تلقائيا

2- ال username وهو VARCHAR(55) ولازم يكون UNIQUE

3- ال phonenum وهو VARCHAR(55) .. هل يمكن أن يكون تليفون من 55 رقم ؟! عادى اهل المريخ بعيدين شوية :D

نحاول ننفذ ال جملة ال SQL لإنشاء الجدول إذا لم يكن موجود ولكن فى حال وجوده سيقدح ل raise Exception ونحن نرشده بإن نطبع رسالة بإن الجدول موجود..

*اضافة مستخدم سيتم عن طريق add_user method المعرفة كالتالى

```
  def add_user(username, phonenum)
    begin
      @db.execute("INSERT INTO users (username, phonenum) VALUES(?,?)",
username, phonenum)
      puts "#{username} is added!"
    rescue Exception => e
      puts e.message
    end
  end
```

عملية insert بسيطة جدا

لاحظ بأنه سيتم استبدل علامات الإستفهام ب المتغيرات username, phonenum

*عرض بيانات مستخدم

```
  def display_user(user)
    puts "-----------------------"
    id=user['id'].to_s
    puts "ID: #{id}"
    puts "Name: #{user['username']}"
    puts "Phone: #{user['phonenum']}"
    puts "-----------------------"
  end
```

لاحظ ان ال user الذي ستأخذه هذه الطريقة عبارة عن row من قاعدة البيانات

ال row متعرف كالتالى

```
id | username | phonenum
```

تقدر توصل لقيمة كل column بالتعامل معاه كأنه hash بما إننا حددنا هكذا فى بداية انشاء ال Connection Object لو تذكر

```
    @db.results_as_hash=true #record['username']
```

*الإستعلام عن مستخدمين

```
  def find_user(username)
    begin
      @db.execute("SELECT * FROM users WHERE username LIKE '#{username}%'") do
|user|
       display_user(user)
      end
    rescue Exception => e
      puts "Not found!"
    end
  end
```

جملة SELECT/LIKE بسيطة أليس كذلك ؟

هنا سيجلب كل ال rows التي فيها ال username column مشابه ل username

*حذف مستخدم

```
  def delete_user(username)
    begin
      delstmt="DELETE FROM users WHERE username='#{username}'"
      @db.execute(delstmt)
      puts "#{username} is deleted."
    rescue Exception => e
      puts e.message
```

```
    end
  end
```

*تحديث بيانات مستخدم

```
  def update_user(username, newphone)
    updatestmt="UPDATE users SET phonenum='#{newphone}' WHERE
username='#{username}'"
    begin
      @db.execute(updatestmt)
      puts "#{username} is updated!"
    rescue Exception => e
      puts e.message
    end
  end
```

*حذف الكل

```
  def delete_all
    delstmt="DROP TABLE users"
    @db.execute(delstmt)
    #create new one
    puts "Table Dropped!"
    create_table
  end
```

انا أرى انه من الاسرع انك تعمل DROP للجدول كله :)

*عرض الكل

```
  def select_all
    selectstmt="SELECT * FROM users"
    begin
      @db.execute(selectstmt) do |rec|
        display_user(rec)
      end
    rescue Exception => e
      puts e.message

    end
  end
```

جملة select بسيطة أليس كذلك ؟

اخيرا كل ما عليك هو استخدام ال class مثلا كالتالى

```
def get_user
  print("Username: ")
  return gets.chomp!
end

def get_phone
  print("Phone: ")
```

```
  return gets.chomp!
end



def entry_point
  plite=PhonerLite.new("mydb")
   mnu=<<M
  1-Add User
  2-Delete User
  3-Update
  4-View All
  5-Clear
  6-Quit

M
  while true
    puts mnu
    opt=gets.chomp.to_i
    if not (1..6).include?(opt)
      puts "Unknown Option"
    end
    if opt==1
      plite.add_user(get_user, get_phone)
    elsif opt==2
      plite.delete_user(get_user)
    elsif opt==3
      plite.update_user(get_user, get_phone)
    elsif opt==4
      plite.select_all
    elsif opt==5
      plite.delete_all
    elsif opt==6
      exit
    end
  end

end

entry_point
```

جميل، مبروك على انهائك التطبيق!

ActiveRecords

انشئ قاعدة بيانات باسم phonerdb_development وفيها جدول كالتالي

```
mysql> describe users;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int(11)      | NO   | PRI | NULL    | auto_increment |
| name      | varchar(55)  | YES  |     | NULL    |                |
| phonenum  | varchar(10)  | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
3 rows in set (0.08 sec)
```

ال ActiveRecords تسمح لك بأنك تعالج الجدول بدون استخدام SQL ولكن بإستخدام ال OO هنا بمجرد أنك تستخدم ActiveRecord سيتم التعامل مع الجدول الخاص بال users عن طريق Ruby) Model (Class

الكلام جميل، ماذا عن الفعل ؟

(شكلك داخل سخن شوية :D)

1- اعمل Import لل active record كالتالي

```
require 'active_record'
```

2- اضبط الإعدادات الخاصة بالإتصال عن طريق ActiveRecord::Base.establish_connection كالتالي

```
ActiveRecord::Base.establish_connection(
  :adapter  => 'mysql',
  :host     => 'localhost',
  :username => 'root',
  :password => PASSWORD_GOES_HERE,
  :database => 'phonerdb_development'

  )
```

هنا حددنا ال

adapter الذي سنعمل به سواء mysql او postgres او غيرها

host وهو يعبر عن مكان الخادم

username، password لإنشاء الإتصال

database قاعدة البيانات التي سيتم الإتصال معاها

تمام، لا ينقص إلا كتابة Model الذي سيتعامل مع ال Table كالتالي

```
#Mapping/Validating.
class User < ActiveRecord::Base
  validates_numericality_of :phonenum
  validates_uniqueness_of   :name
  validates_presence_of     :name
  validates_presence_of     :phonenum

end
```

فقط!

أين اسم ال table ؟!

همممم الإجابة ان Ruby/ActiveRecords ذكيين كفاية ليعرفوا أن الجدول هو جمع User يعني users

Conventions over Configurations

تستطيع أن تحدد الجدول طبعا يدويا باستخدام set_table_name كالتالي مثلا

```
class User < ActiveRecord::Base
  set_table_name("users")
  validates_numericality_of :phonenum
  validates_uniqueness_of   :name
  validates_presence_of     :name
  validates_presence_of     :phonenum

end
```

جيد، تعالى نشوف السحر الذي أخبرتك عنه !!

***اضافة user**

```
def add_user(username, phone)
  user=User.new
  user.name=username
  user.phonenum=phone
  user.save! #throws an exception.
end
```

كل ما عليك هو انك تنشئ object من ال User class وتحدد قيمته في كل عمود، فى مثالنا هنا ال name وال phonenum

وتعمل save! بدل من insert statement أليس هذا أسهل ؟

***عرض بيانات user**

```
def display_user(user)
  puts "--------------------"
  puts "ID: #{user.id.to_s}"
  puts "Name: #{user.name}"
  puts "Phone: #{user.phonenum}"
  puts "--------------------"
end
```

***البحث**

```
def find_user(username)
  #Limit to 1.
  user=User.find(:first, :conditions =>["name=?", username])
  #u may use User.find_by_name method. god bless reflection :)
```

```
  #user=User.find_by_name(username)


  return user
end
```

هنا نحتاج لوقفة، انت تستطيع أن تستخدم

```
  user=User.find(:first, :conditions =>["name=?", username])
```

هنا سيرجع أول سجل سيقابله بحيث ان ال name يساوى ال username المطلوب

وتستطيع أيضا أن تستخدم ميزة رائعة

```
user=User.find_by_name(username)
```

كيف عرفت أن اسم column بإسم name ؟ هممم راجع ال reflection :)

*تحديث مستخدم

```
def update_user(username, newphone)
  user=find_user(username)
  user.phonenum=newphone
  user.save!
end
```

ستحصل على ال object المطلوب بإستخدام عملية find بإستخدام الدالة التي كتبناها find_user وتعدل عليه وتعمل save! وتم الحفظ فى الجدول :D

*عرض الكل

```
def select_all
  User.find(:all){|user|
    yield user
  }
end
```

هنا لن يتم العرض ولكن سيحصل yield كل record لاستخدامه فى display_user بصورة مناسبة

*حذف

```
def delete_user(username)
  user=find_user(username)
  user.destroy
  puts "Deleted!"
end
```

احصل على object واستدعى ال destroy method الخاصة به

حذف الكل*

```
def delete_all
  User.delete_all
end
```

تستطيع أن تحدد conditions طبعا فى عملية الحذف الجماعى :)

اخيرا استغل كل مافى ال module كالتالى مثلا

```
def get_user
  print("Username: ")
  return gets.chomp!
end

def get_phone
  print("Phone: ")
  return gets.chomp!
end



def entry_point

   mnu=<<M
  1-Add User
  2-Delete User
  3-Update
  4-View All
  5-Clear
  6-Quit

M
  while true
    puts mnu
    opt=gets.chomp.to_i
    if not (1..6).include?(opt)
      puts "Unknown Option"
    end
    if opt==1
      add_user(get_user, get_phone)
    elsif opt==2
      delete_user(get_user)
    elsif opt==3
      update_user(get_user, get_phone)
    elsif opt==4
      for user in select_all
        display_user(user)
      end
    elsif opt==5
      delete_all
    elsif opt==6
      exit
    end
  end
```

```
end

entry_point
```

(وبس كدا :D)

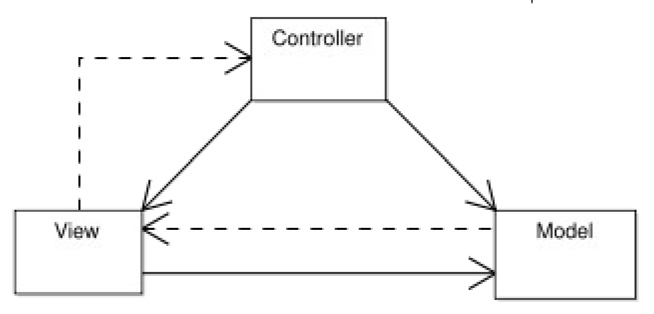
الفصل الرابع عشر: Ruby on Rails

لفتت Ruby الإنتباه فى الفترة الأخيرة بصورة كبيرة وذلك بسبب Rails وهى كما يعلم العديد انها Web Framework تطبق مبدأ ال MVC بسلاسة متكاملة.. هدفها الأول هو السرعة والبساطة والكفاءة والأمان! (كل شئ تحتاجه أليس كذلك ؟) وتراعى مبدأ DRY (Don't Repeat Yourself) وهو من المفترض أن يكون شعار الجميع! تقدم لك scaffolding (ميزة داخلية) بمعنى انك تستطيع أن تعمل manipulate لل models (ال Database Tables) من خلال العمليات الأساسية CRUD و تولد ال views الخاصة بهم وكل هذا من خلال ضغطة زر أو أمر مبسط.

crud: (**c**reate **r**ead **u**pdate **d**elete)

تعالى نتكلم عن ال *MVC* قليلا

ال MVC هو نموذج pattern يعتمد عليه كثيرا فى هندسة البرمجيات وفيه تفصل ال العرض (view) عن المتحكم عن ال MODEL ، يمكن أن الفكرة لم تصلك فتعالى نوضحها بمثال:



المستخدم يبعث طلب للمتحكم مثلا بأن يضيف كتاب

amazon/books/add
domain/controller/action

هنا ال controller هو ال books

وال action هو add

ال books يشبه المقسم هو الذي يعطي تعليمات للقاعدة البيانات أو ال model لنكون واضحين أكثر على فرض ان عندنا جدول بإسم books فى قاعدة البيانات .. الجدول هذا المفترض انك أن تستطيع أن تضيف كتاب جديد وتعدل كتاب موجود او تحذف، وهكذا بالضبط الذي يحصل، حيث يصل الأمر المطلوب "تعديل اضافة ازاله" للمتحكم وهو ينفذه على ال model بكل بساطة!

فى مثالنا هذا أرسل طلب لـ books controller ب action هو add بحيث انه يوجه المستخدم ل view بسيطة تسمح له بأن يدخل بيانات الكتاب ويعتمده فى قاعدة البيانات!

مثال اخر:

amazon/books/edit/12
domain/controller/action/param

هنا فى المثال ال action هو edit "سنعمل تعديل لكتاب موجود" ولكن اى كتاب ؟ الإجابة هو الكتاب صاحب id = 12 ! فيطلع لك view بسيطة تقدر تعدل وتعتمد الكتاب!

تقدر تنفذ كل هذا بمجرد انك تنشئ scaffold مثلما سترى فى التطبيق القادم.

بعض الملاحظات :

Rails تقسم لعدة أجزاء منها مثلا ال Controller ويعطيك القدرة -مثلما قلنا - على التعامل مع ال actions المطلوبة بكل سهولة.

فى ActiveRecord وهو عبارة عن ORM (Object/Relational Mapping) يقوم بعمل مثل الخريطة بين ال model و table يخزن فيه الداتا الخاصة بال model ، بمعنى لن تشغل بالك بنوعية قاعدة البيانات التي تعمل عليها لأنه و بكل بساطة ال ActiveRecord سيحل لك كل مشاكلك!

ال Database Migrations: تسمحلك بإجراء التغييرات مثلا اضافة جدول او اضافة عمود او تغير اسمائهم وغيرها!

كلمة model معناها انه يوجد جدول فى قاعدة البيانات مكافئ لها فى ال ActiveRecord

هيكيلة تطبيق ***Rails***

```
├───app
│   ├───controllers
│   ├───helpers
│   ├───models
│   └───views
│       ├───layouts
│       └───people
├───config
│   ├───environments
│   └───initializers
├───db
│   └───migrate
├───doc
├───lib
│   └───tasks
├───log
├───nbproject
│   └───private
├───public
│   ├───images
│   ├───javascripts
│   └───stylesheets
├───script
│   ├───performance
│   └───process
├───spec
├───test
│   ├───fixtures
│   ├───functional
│   ├───integration
│   ├───mocks
│   │   ├───development
│   │   └───test
│   └───unit
├───tmp
│   ├───cache
│   ├───pids
│   ├───sessions
│   └───sockets
└───vendor
    └───plugins
```

التطبيق يقسم فى داخله لكذا قسم منها -الذي يهمنا حاليا- هو ال app ويضم ال controllers وال models وال views بالإضاف إلى ال helper classes

ال config يشمل ال Configurations الخاصة بال Applications مثل ال database.yml وفيها يحدد بيانات الدخول ونوع القاعدة البيانات.

ويكون فيها ال routes التى تحدد ال mapping بين ال url وال controller

ال db و يشمل مجلد migrate وفيه يخزن ال migrations كلها

ال public يخزن فيه الحاجات العامة مثل style sheets, jsS, images,.. etc

لمعرفة المزيد عن نشأة Rails وفلسفتها راجع
http://en.wikipedia.org/wiki/Ruby_on_Rails

:Rake

هى أداة مشابهة ل make فى UNIX/UNIX-Like

تقدم لك امكانية تنفيذ عدد معرف مسبقا من المهام

للإطلاع على المهام
http://www.tutorialspoint.com/ruby-on-rails/rails-and-rake.htm

:WebRick

هو HttpServer library مكتوب ب Ruby
وتستطيع الإطلاع على معلومات اكثر بخصوصه هنا
http://www.webrick.org

HolaRails

اولا: قم بتثبيت rails

```
gem install rails --include-dependencies
```

كالعادة سنبدأ ب Hello World
من ال commandline او ال terminal اكتب

```
rails holaRails
```

وهنا قمنا بعمل تطبيق جديد بإستخدام rails فقامت rails بإنشاء ال skeleton للتطبيق مثلما تكلمنا فى هيكلية تطبيق RoR

```
C:\ruby\rProjects>rails holaRails
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      create  config/environments
      create  config/initializers
      create  db
      create  doc
      create  lib
      create  lib/tasks
      create  log
      create  public/images
      create  public/javascripts
      create  public/stylesheets
      create  script/performance
      create  script/process
      create  test/fixtures
      create  test/functional
      create  test/integration
      create  test/mocks/development
      create  test/mocks/test
```

```
      create  test/unit
      create  vendor
      create  vendor/plugins
      create  tmp/sessions
      create  tmp/sockets
      create  tmp/cache
      create  tmp/pids
      create  Rakefile
      create  README
      create  app/controllers/application.rb
      create  app/helpers/application_helper.rb
      create  test/test_helper.rb
      create  config/database.yml
      create  config/routes.rb
      create  public/.htaccess
      create  config/initializers/inflections.rb
      create  config/initializers/mime_types.rb
      create  config/boot.rb
      create  config/environment.rb
      create  config/environments/production.rb
      create  config/environments/development.rb
      create  config/environments/test.rb
      create  script/about
      create  script/console
      create  script/destroy
      create  script/generate
      create  script/performance/benchmarker
      create  script/performance/profiler
      create  script/performance/request
      create  script/process/reaper
      create  script/process/spawner
      create  script/process/inspector
      create  script/runner
      create  script/server
      create  script/plugin
      create  public/dispatch.rb
      create  public/dispatch.cgi
      create  public/dispatch.fcgi
      create  public/404.html
      create  public/422.html
      create  public/500.html
      create  public/index.html
      create  public/favicon.ico
      create  public/robots.txt
      create  public/images/rails.png
      create  public/javascripts/prototype.js
      create  public/javascripts/effects.js
      create  public/javascripts/dragdrop.js
      create  public/javascripts/controls.js
      create  public/javascripts/application.js
      create  doc/README_FOR_APP
      create  log/server.log
      create  log/production.log
      create  log/development.log
      create  log/test.log
```

جميل شغل webrick كالتالى

```
ruby script/server
```

--ملحوظة ال WebRick سيستخدم بورت 3000 إلا فى حال إذا حددت غير هذا بإستخدم p option-

كالتالى مثلا :

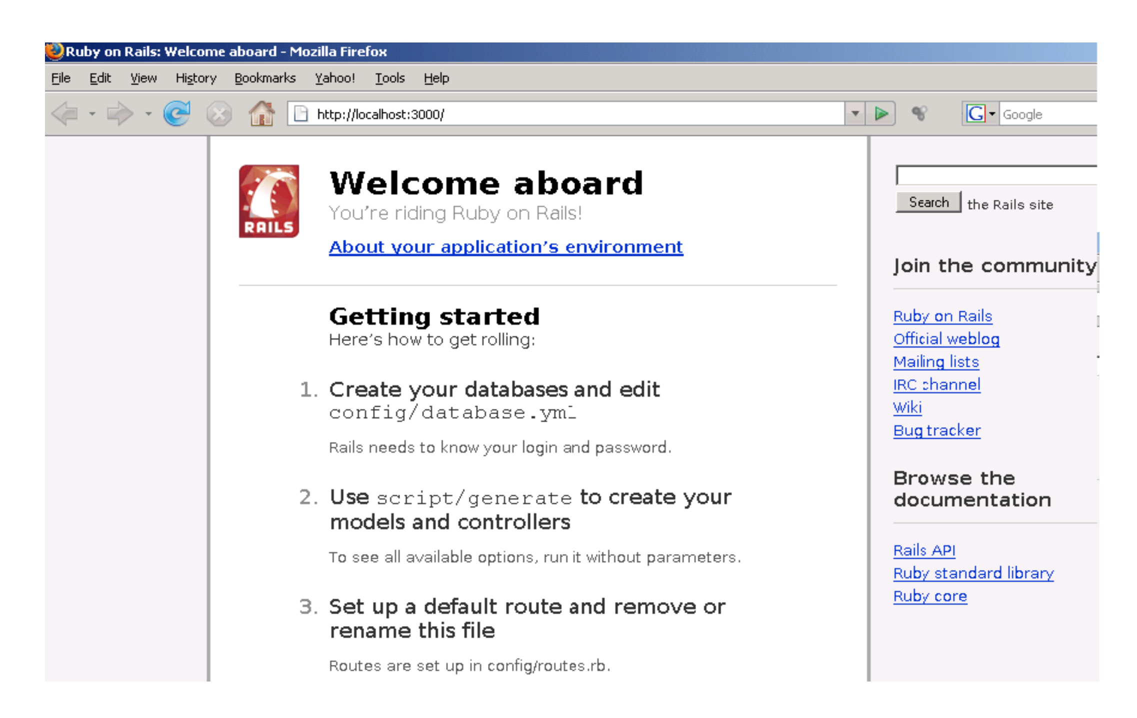
ruby script/server -p 9000

هكذا سيعمل على بورت 9000 على كل حال خلينا مع ال Default

```
C:\ruby\rProjects\holaRails>ruby script/server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2008-05-13 01:47:02] INFO  WEBrick 1.3.1
[2008-05-13 01:47:02] INFO  ruby 1.8.6 (2007-09-24) [i386-mswin32]
[2008-05-13 01:47:02] INFO  WEBrick::HTTPServer#start: pid=2448 port=3000
```

افتح المتصفح واكتب فى العنوان

```
localhost:3000
```



رائع ، لم يتبق إلا أن نكتب التطبيق

1- سنعمل Controller جديد ليتحكم فى التعامل مع التالى

```
localhost/say/hello
```

تعرض لينا رسالة Hello World مثلا

و

```
localhost/say/goodbye
```

يعرض لنا رسالة Good Bye

```
localhost/say/time
```

يعرض لنا ال time

ال Controller هنا بإسم say وله action بإسم hello

استخدم script/generate controller لتنفيذ الذي اتفقنا عليه كالتالى

```
C:\ruby\rProjects\holaRails>ruby script/generate controller say hello goodbye
time
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/say
      exists  test/functional/
      create  app/controllers/say_controller.rb
      create  test/functional/say_controller_test.rb
      create  app/helpers/say_helper.rb
      create  app/views/say/hello.html.erb
      create  app/views/say/goodbye.html.erb
      create  app/views/say/time.html.erb
```

كدا أنشأ متحكم بإسم say وانشئت ال views الخاصة بكل action "الذي ستعرض فيها كل action".

افتح ملف app/controllers/say_controller.rb في ال Editor المفضل لديك ستجده ملف ruby عادى جدا كالتالى :

```
class SayController < ApplicationController

  def hello
  end

  def goodbye
  end

  def time
  end
end
```

سنلاحظ ان أي متحكم يورث من ال ApplicationController class والطرق التي يحويها هى ال Actions خاصة به.

طيب جميل نريد أن نجرب .. افتح ال Browser واكتب localhost:3000/say/hello

ستظهر لك

```
Say#hello
Find me in app/views/say/hello.html.erb
```

اذا ظهرت لك رسالة error مشابهه ل

```
 no such file to load -- sqlite3
```

فلازم تثبت sqlite3, sqlite3-ruby لأنها اصبحت ال Default DB فى rails بسبب انها مناسبة جدا لبيئة التطوير

هكذا ال action أصبح يعمل بشكل جيد، لم يبق إلا أن نعدل في ال view او الصفحة التي ستظهر للمستخدم

عدل على الملف app/views/say/hello.html.erb

بما يناسبك كالتالى مثلا

```
<html>
   <head>
      <title>Hola!</title>
   </head>
   <body>
       <center><h1>Hello World!</h1></center>
   </body>
</html>
```

Hola! - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

http://localhost:3000/say/hello

Hello World!

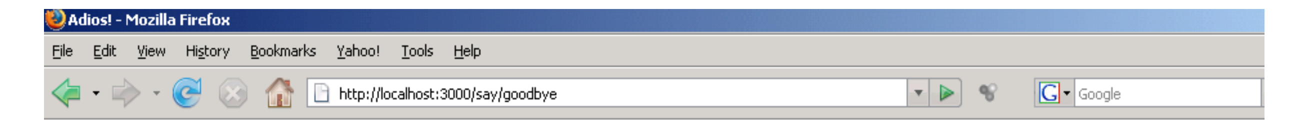
تمام جدا لم يبق إلا أن تعدل في ال views لكل action التي تريدها مثل goodbye و time

اعمل edit ل app/views/say/goodbye.erb.html

```
<html>
   <head>
      <title>Adios!</title>
   </head>
   <body>
       <center><h1>Good Bye!</h1></center>
   </body>
</html>
```

واعمل save وجرب تفتح

```
localhost:3000/say/goodbye
```



Good Bye!

لم يتبق إلا time بس هممممم كيف نظهر ال time ؟

تماما اعمل edit لل view في app/view/time.erb.html كالتالى

```
<html>
   <head>
      <title>Time View</title>
   </head>
   <body>
         <center><h1>Now: <%=@now%> </h1></center>
   </body>
</html>
```

ما هذا ؟!!

مجرد html عادى ولكن فيه ruby code

```
<%=@now%>
```

هنا سيتم اعادة قيمة المتغير now ، لكن لحظة! أين عرف هذا المتغير اصلا؟ هذه اول مرة يظهر فيها، بالفعل كلامك صحيح، تعالى نعرف المتغير كالتالى.. اعمل edit لل SayController كالتالى :

```
class SayController < ApplicationController

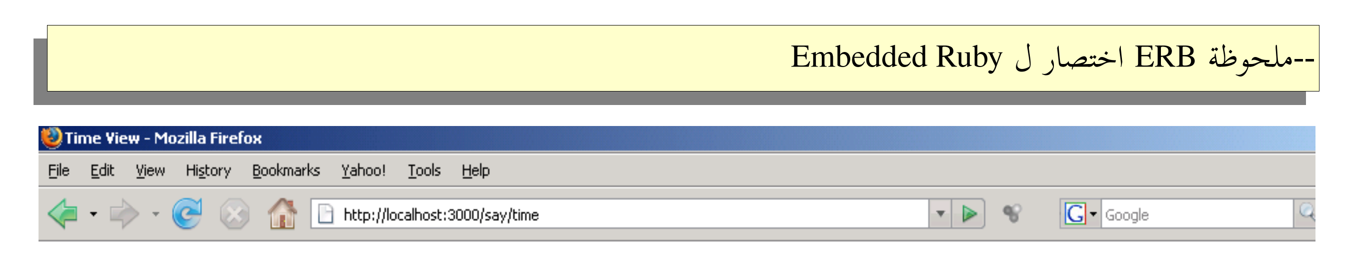
  def hello
  end

  def goodbye
  end

  def time
      @now=DateTime.now.to_s #string repr!
  end
end
```

لاحظ اننا عرفنا المتغير now فى ال time action وبهذه الطريقة تستطيع أن تستخدمه فى اى ruby code وتحديدا ال ERB

--ملحوظة ERB اختصار ل Embedded Ruby



Now: 2008-05-13T02:35:57+03:00

نظرة سريعة على ال database.yml

فى نقاشنا السابق لم نستخدم أي Database Engine لأن الموضوع لم يتطلب ذلك، ولكن لابد من الكلام عنها

افتح ملف config/database.yml بأى محرر

```
# SQLite version 3.x
#   gem install sqlite3-ruby (not necessary on OS X Leopard)
development:
  adapter: sqlite3
  database: db/development.sqlite3
  timeout: 5000

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: sqlite3
  database: db/test.sqlite3
  timeout: 5000

production:
  adapter: sqlite3
  database: db/production.sqlite3
  timeout: 5000
```

ستلاحظ

1- ان ال adapter الإفتراضى هو sqlite3 اذا أحببت ان تغيره تستطيع أن تعدله بإستخدام -d وتحدد ال db engine مثال على استخدام mysql

```
rails myapp -d mysql
```

2- توجد ثلاث قواعد بيانات للتطوير والإختبار واخيرا عملية الإنتاج

3- الملف مكتوب ب yml syntax وهو اسلوب فى كتاب ال data مشابه ل xml ولكنه مش markup language ويمكن يكون اشهر من ال xml فى حياة مبرمجين ruby :)

هل فى parsers لل yml ؟

اكيد بالطبع وليس لـ ruby فقط .. للمزيد من المعلومات
/http://www.yaml.org

Rails 2 Minutes PhoneBook

تطبيقنا سيكون عبارة عن PhoneBook متكامل يتم فيه تسجيل اسم ورقم تليفون مستخدم معين في قاعدة بيانات

المتطلبات:

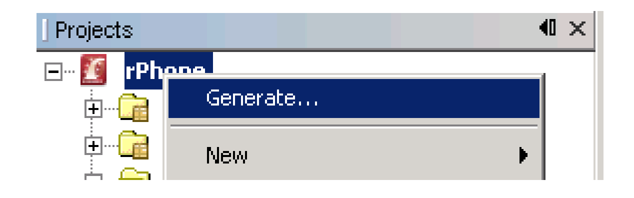
NetBeans 6.1
MySQL Server (اذا استخدمت محرك قاعدة بيانات آخر مثل SQLite او غيرها قم بتعديل ال database.yml بما يناسبك)

File -> New Project -> Ruby -> Ruby on Rails Application

حدد ال Project Name ب rPhone مثلا

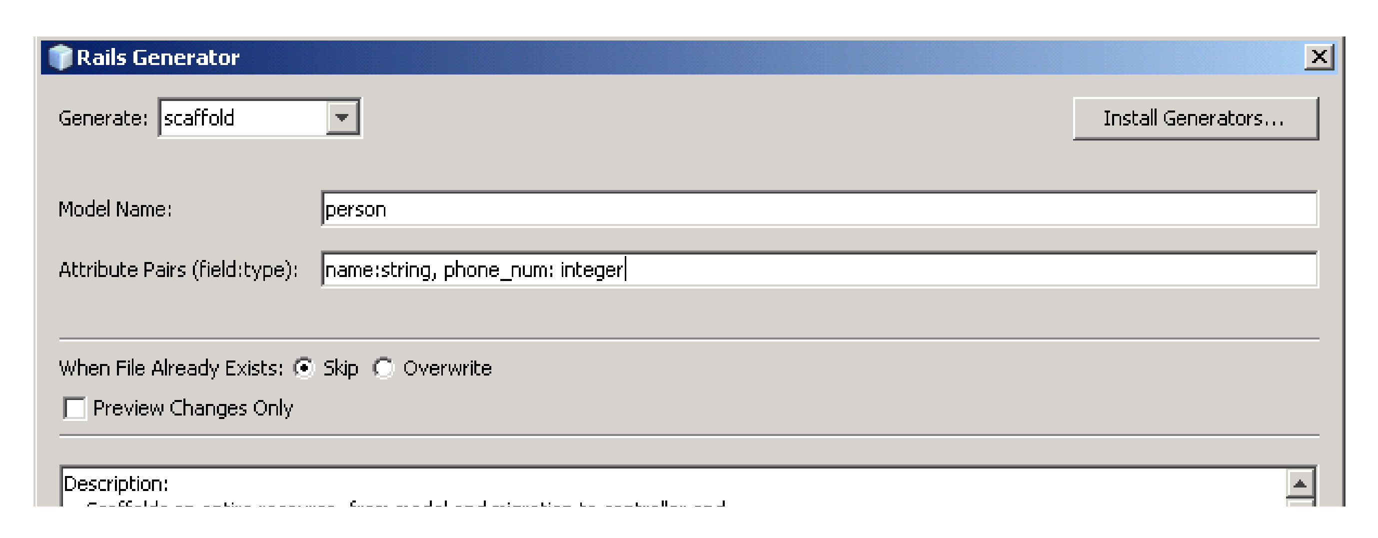
في ال Development Database اعمل Create DB وأعطيها الإسم rphonebookdb
كذا تم إن شاء ال Skeleton الخاص بتطبيقنا

اضغط R.Click على rPhone



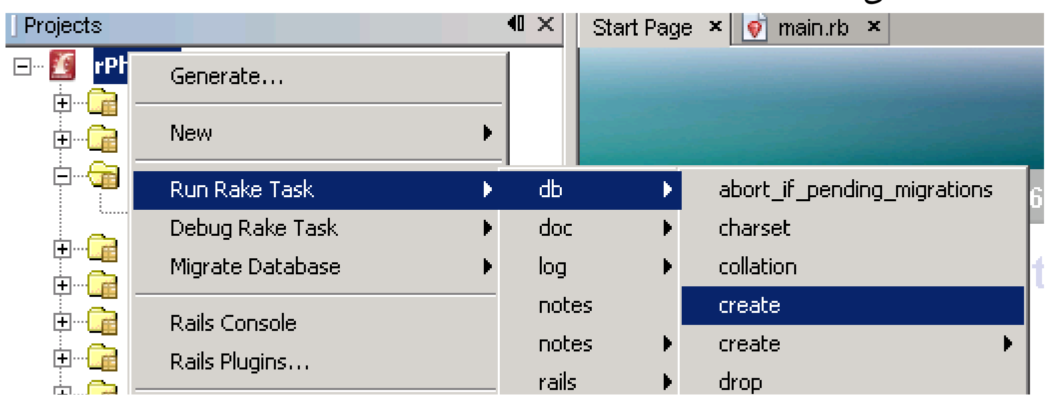
ومنها اختار Generate

اختار scaffold كما بالصورة وحدد اسم ال model ب person
لاحظ ال Attribute Pairs هنا فيها ال name من النوع string و phone_num من النوع integer كل ماعليك هو انك تفصل بين كل field وال type ب :



هنا سيتم انشاء ال model (person) وسيتم انشاء ال Controller الخاص به + عمليات CRUD الأساسية

انشئ قاعدة البيانات كالتالى



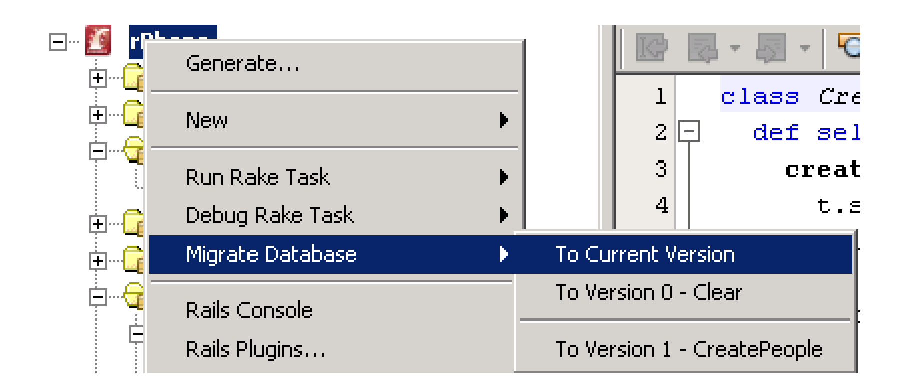
R.Click on rPhone-> Run Rake Task -> db -> create

تمام جدا الآن سنحدد ال Columns التى فى ال table

ملحوظة اى model تعمله يجب أن يكون له table مقابل له، فكل الذي عليك ان تفتح ال DB Migrations كالتالى واختار منها create_people.rb اذا أردت أن تعدل فيه شيء .. إذا لا ادخل على الخطوة التالية



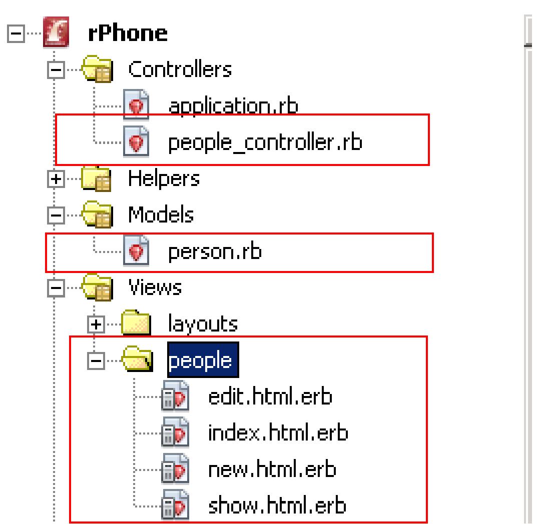
الآن نستطيع أن نعمل migrate ونحن مطمئنين كالتالي



وبس، مبروك!

نعم ؟ أين الأكواد وأين وأين ..!

ال model وال controller وال views كلها أنشأت بمجرد أن عملنا Generate ل Scaffold، اذا لم تصدقني افتح ال controllers folder وستجد فيه people_controller.rb وفيه العمليات الأساسية التي يقدمها ال scaffold ومايقابلها من ال views



اخيرا شغل برنامجنا الطويل والمتعب والمرهق الذى اخد دقيقتين

Run -> Run Main Project (F6)



Listing people

Name	Phone num			
Ahmed Youssef	0127541412	Show	Edit	Destroy
Ahmed Mostafa	0123654789	Show	Edit	Destroy
Christina	0108752612	Show	Edit	Destroy
Ahmed Omar	0124147121	Show	Edit	Destroy
Marian	0257814787	Show	Edit	Destroy
salma	0125478961	Show	Edit	Destroy

New person

لإضافة record جديد اضغط New person



New person

Name

Rogina

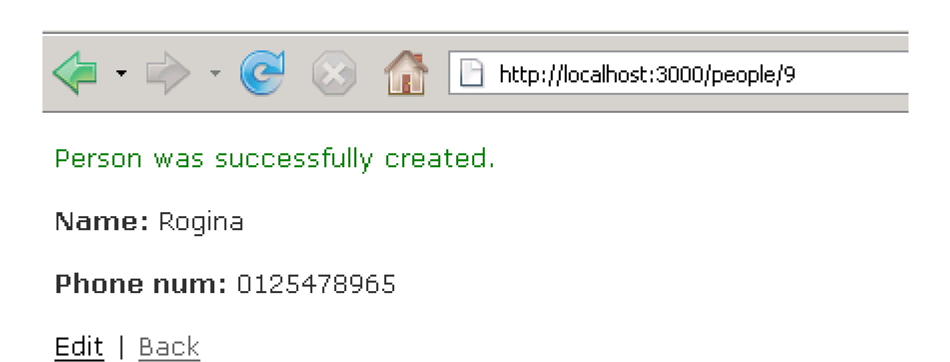
Phone num

0125478965

Create

Back

اضغط create



تستطيع أن تحرر Edit كالتالى مثلا

http://localhost:3000/people/9/edit

Editing person

Name

Rogina

Phone num

0125478965

Update

Show | Back

واضغط Update

http://localhost:3000/people/9

Person was successfully updated.

Name: Rogina

Phone num: 0125478962

Edit | Back

اخيرا كل ماعلينا هو وضع بعض القوانين لضمان سلامة البينات التى سيتم ادخالها لقاعدة البيانات زى طول الإسم + ان رقم التليفون لازم يكون رقم

جميل، سنضيف هذا في ال person model

```
class Person < ActiveRecord::Base
```

```
  validates_length_of :name, :within => 2..20
  validates_length_of :phone_num, :is => 10

  validates_numericality_of :phone_num
  validates_uniqueness_of :name

end
```

هنا سنجبره ان يعمل فحص على طول الإسم بحيث انه يكون بين 2 و 20 بالكثير وسنحدد عدد ارقام التليفون ب 10 ارقام (تقدر تعملها within مثل سابقتها وتحدد اقل رقم واعلى رقم) بإستخدم validates_length_of

نريده أن يفحص ان كل الذي سيتم ادخاله فى phone_num يكون ارقام فقط فسنستخدم validates_numericality_of

ونريد أن يكون اسم المستخدم فريد دائما فسنستخدم validates_uniqueness_of

لاحظ أنه ليس هذه كل ال validators التي تستطيع أن تستخدمها.. لكن تستطيع أن تكتب validator خاص بك أيضا.

جيد تعالى نجرب نضيف سجل بيانات غير سليمة ونرى كالتالى

http://localhost:3000/people/new

New person

Name

Nour

Phone num

012aa

Create

Back

سيظهر لنا التالى

New person

2 errors prohibited this person from being saved

There were problems with the following fields:

- Phone num is the wrong length (should be 10 characters)
- Phone num is not a number

Name

Nour

Phone num

012aa

Create

Back

اذا كنت من النوع الذي يحب أن يستخدم ال command-line/terminal كثير ومجرد text editor تستطيع أن تراجع ال commands الخاصة بتشغيل الويب سرفر وال generators هنا
http://www.tutorialspoint.com/ruby-on-rails/rails-quick-guide.htm

ماذا الآن؟

الطريق أمامك مفتوح فى مجالات كثيرة !!
تستطيع أن تتجه لل GUI من خلال wxRuby او FXRuby
أو تطوير الويب من خلال ROR
أو برمجة شبكات من خلال مكتبات الشبكات الأكثر من رائعة المتوفرة فى Ruby
أو كتابة Extensions ل Ruby من خلال ال C مثلا
أو تستطيع أن تتجه إلى Jruby "وهى 100% بال Java"
أو تستطيع أن تتجه الى IronRuby وتستفيد فيها من قوة ال .NET
تابع مشاريع مفتوحة المصدر مكتوبة ب Ruby على RubyForge
كل المصادر المتعلقة ب ruby ستجدها فى RubyMatters

كتب انصح بها

Apress Beginning Ruby
Design Patterns in Ruby - Addison Wesley
The Ruby Programming Language
Agile Web Development with Rails

لقد فعلتها !!

مبروك على انتهائك من الكتاب

شكر

احب ان اقوم بتوجيه شكر **خاص** ل
Christina: الكتاب عمره ماكان هيخلص فى وقت قياسى بدون مساعدتك
Squall: مساندتك ليا دائما فخر اعتز بيه
شكرا ل OMLX و وادى التقنية على التنسيق والمراجعة ومجهودهم الرائع

شكر

شكرا SAMI على الغلاف الأكثر من رائع

شكرا لكل من

St0rM, BlackRay, sAFA7_eLNeT, Muslim, Binary, Dj.Raidy, GreyHunter, HackoBacko, ,MySQL